UNIVERSIDADE FEDERAL FLUMINENSE

UÉVERTON DOS SANTOS SOUZA

Multivariate Investigation of NP-Hard Problems: Boundaries Between Parameterized Tractability and Intractability

> NITERÓI 2014

UNIVERSIDADE FEDERAL FLUMINENSE

UÉVERTON DOS SANTOS SOUZA

Multivariate Investigation of NP-Hard Problems: Boundaries Between Parameterized Tractability and Intractability

Thesis presented to the Computing Graduate Program of the Universidade Federal Fluminense in partial fulfillment of the requirements for the degree of Doctor of Science.

Advisors: Fábio Protti Maise Dantas da Silva Dieter Rautenbach

> NITERÓI 2014

Multivariate Investigation of NP-Hard Problems: Boundaries Between Parameterized Tractability and Intractability

Uéverton dos Santos Souza

Thesis presented to the Computing Graduate Program of the Universidade Federal Fluminense in partial fulfillment of the requirements for the degree of Doctor of Science.

Aproved by:

Prof. Fábio Protti / IC-UFF (Presidente)

Prof. Maise Dantas da Silva / RCM-UFF

Prof. Carlos Alberto de Jesus Martinhon / IC-UFF

Prof. Jayme Luiz Szwarcfiter / NCE-UFRJ

Prof. Sulamita Klein / DCC-UFRJ

Prof. Vinícius Gusmão Pereira de Sá / DCC-UFRJ

Commit your way to the Lord; Trust him, and he will act. (PSALMS 37:5)

To my mother, Iracemar dos Santos Souza.

Acknowledgments

I would like to express my gratitude to my supervisor, Professor Fábio Protti, for all his patience and all the opportunities he gives me every day to grow as a researcher and as a person. I also want to thank my co-advisor Professor Maise Dantas da Silva for all his technical knowledge that he shared with me since the beginning of our cooperation. I also want to express my gratitude to Professor Michael Fellows and the committee members for the important contributions and suggestions that helped to improve the quality of this thesis. I would like to express, in particular, my special gratitude to Professors Dieter Rautenbach and Lucia Draque Penso for accepting me as an external researcher at Universität Ulm for four months in Ulm, Germany.

I would like to thank my family for the unconditional love and everything you have done for me.

I also would like to thank my friends and the members of CEDERJ, ITR-UFRRJ, and IC-UFF, students, employees, and professors, for the support, the infra-structure, the companionship, and the advices.

Finally, I would like to acknowledge CAPES and DAAD for financial support.

Resumo

O principal objetivo ao utilizar computação para solucionar um problema é desenvolver um mecanismo para solucioná-lo de forma eficiente. Em geral, essa eficiência é associada com solucionabilidade em tempo polinomial. A teoria de NP-completude foi desenvolvida para mostrar quais problemas provavelmente não possuem algoritmos de tempo polinomial para solucioná-los. No entanto, muitos problemas NP-difíceis e NP-completos ainda devem ser solucionados na prática, sendo assim natural perguntar se estes problemas admitem um algoritmo cuja complexidade de tempo não-polinomial seja puramente uma função de algum subconjunto de seus aspectos. Questões sobre a existência de tais algoritmos são tratadas pela teoria da complexidade computacional parametrizada, desenvolvida por Downey e Fellows.

Nesta tese, apresentamos uma investigação multivariada da complexidade de alguns problemas NP-difíceis. Primeiro, fazemos uma análise sistemática da complexidade desses problemas, analisando seus subproblemas e mapeando quais destes pertencem a cada um dos lados de uma "fronteira imaginária" entre tratabilidade e intratabilidade (considerando-se como medida de eficiência a complexidade polinomial). Em seguida, analisamos quais conjuntos de aspectos destes problemas são *fontes* de intratabilidade, ou seja, conjuntos de aspectos para os quais existe um algoritmo para resolver o problema associado cuja complexidade de tempo não-polinomial é puramente uma função desses conjuntos. Desta forma, usamos complexidade computacional clássica e parametrizada de forma alternante e complementar, para mostrar quais subproblemas dos problemas apresentados são NP-difíceis, e por último para diagnosticar para quais conjuntos de parâmetros (aspectos encapsulados) os problemas são tratáveis por parâmetros fixos.

Esta tese exibe uma análise clássica e parametrizada de diferentes grupos de problemas NP-difíceis. Os problemas abordados estão divididos em quatro grupos de natureza distinta, no contexto de estruturas de dados, jogos combinatórios e teoria dos grafos: (I) solução para grafos e/ou e suas variantes; (II) jogos de inundação; (III) problemas relativos à convexidade P_3 em grafos; (IV) problemas sobre emparelhamento induzido em grafos.

Palavras-Chave

- 1. Complexidade Computacional
- 2. Tratabilidade por Parâmetro Fixo
- 3. Método de Redução a um Núcleo
- 4. Intratabilidade Parametrizada
- 5. Solução para Grafos E/Ou
- 6. Inundação em Grafos
- 7. Convexidade P_3
- 8. Emparelhamento Induzido

Abstract

The main goal when using computing to solve a problem is to develop a mechanism to solve it efficiently. In general, this efficiency is associated with solvability in polynomial time. The theory of NP-completeness was developed to show which problems probably do not have polynomial time algorithms. However, many NP-hard and NP-complete problems must still be solved in practice; therefore it is natural to ask if each of these problems admits an algorithm whose non-polynomial time complexity is purely a function of some subset of its aspects. Questions about the existence of such algorithms are addressed within the theory of parameterized computational complexity developed by Downey and Fellows.

In this thesis we present a multivariate investigation of the complexity of some NPhard problems, i.e., we first develop a systematic complexity analysis of these problems, defining its subproblems and mapping which one belongs to each side of an "imaginary boundary" between polynomial time solvability and intractability. After that, we analyze which sets of aspects of these problems are *sources* of their intractability, that is, subsets of aspects for which there exists an algorithm to solve the associated problem, whose nonpolynomial time complexity is purely a function of those sets. Thus, we use classical and parameterized computational complexity in an alternate and complementary approach, to show which subproblems of the given problems are NP-hard and latter to diagnose for which sets of parameters the problems are fixed-parameter tractable, or in FPT.

This thesis exhibits a classical and parameterized complexity analysis of different groups of NP-hard problems. The addressed problems are divided into four groups of distinct nature, in the context of data structures, combinatorial games, and graph theory: (I) *and/or* graph solution and its variants; (II) flooding-filling games; (III) problems on P_3 -convexity; (IV) problems on induced matchings.

Keywords

- 1. Computational Complexity
- 2. Fixed-Parameter Tractability
- 3. Kernelization
- 4. Parameterized Intractability
- 5. And/Or Graph Solution
- 6. Flood-filling Games on Graphs
- 7. P_3 -Convexity
- 8. Induced Matching

Contents

1	Intr	roduction	1
	1.1	Background	2
		1.1.1 Parameterized Tractability in Pratice	3
		1.1.2 Multivariate Investigation	4
	1.2	Organization of this Thesis	8
2	Par	ameterized Complexity	12
	2.1	Bounded Search Tree Technique	13
	2.2	Kernelization	14
	2.3	Parameterized Intractability	16
		2.3.1 Analog of Cook's Theorem	17
	2.4	Infeasibility of Polynomial Kernels	18
3	Cor	nplexity of And/Or Graph Solution	21
-	3.1	NP-hardness Results	25
	3.2	Parameterized Complexity Results	27
	3.3	Infeasibility of Polynomial Kernels	37
	3.4	Open Problem	38
4	Flo	oding Graphs	39
	4.1	Flood-it on Trees	43
		4.1.1 Analogous Problems	45
		4.1.2 Phylogenetic Colored Trees	48
		4.1.3 Weighted-Flood-it	51
		4.1.4 Flood-it on 3-colored Trees	51
		4.1.5 Multi-Flood-it on Trees	52
		4.1.6 Free-Flood-it on Trees	54
	4.2	Flood-Filling Games on Power Graphs	56
		4.2.1 Flood-it on Circular Boards	57
		4.2.2 Free-Flood-it on Powers of Cycles	60
	4.3	The Size of a Minimum Vertex Cover as Parameter	65
		4.3.1 Polynomial Kernelization	67
	4.4	Final Considerations	68
		4.4.1 Open Problems	69
5	On	P_3 -convexity	71
		5.0.2 Planar SAT-AM3	72
	5.1	P_3 -Hull Set	74

	5.2 5.3	P_3 -Geodetic Set<	82 84 85	
6	Indu 6 1	aced Matchings close to Maximum Matchings	86 87	
	$6.1 \\ 6.2$	Graphs G with $\nu(G) = \nu_2(G)$	87 90	
		6.2.1 Structure	90 01	
	6.3	Open Problem Open And And And And And And And And And An	91 96	
7	Con	clusions	97	
Bibliography				

List of Figures

1.1	One possible state of knowledge about subproblems of an NP-complete	
	problem Π . An arrow from Π^1 to Π^2 signifies that Π^1 is a subproblem of	
	$\Pi^2.$	5
3.1	(a) A weighted and/or graph; (b) A weighted x-y graph	22
3.2	A graph G and the corresponding and/or graph G'	26
3.3	(a) A graph G, (b) the corresponding instance of MIN-AND/OR ⁰ (k)	29
3.4	FPT-reduction of graph Q in (a) to x-y graph G in (b)	31
3.5	Example of application of Rule 4a	32
3.6	Example of application of Rule 4b	33
3.7	Example of application of Rule 4c	34
3.8	Example of application of Rule 4d	35
3.9	Example of application of Rule 6	36
4.1	An optimal sequence of moves to flood a 3×3 grid	40
4.2	(a) A graph G ; (b) tree T obtained from G	44
4.3	Cpc-tree T' obtained from the pc-tree presented in Figure 4.2(b)	50
4.4	(a) A $2 \times n d$ -board B. (b) A subgraph of the hypergraph H of B	58
4.5	(a) $2 \times n$ circular grid for even n ; (b) $2 \times n$ circular grid for odd n	60
4.6	(a) an edge e_i ; (b) gadget corresponding to e_i	61
4.7	(a) a graph G ; (b) the graph Q ; (c) the graph C_n^2 obtained from G_Q	62
4.8	A sequence of moves to flood the arms of the gadget presented in Fig. $4(b)$.	63
4.9	States of the graph in Figure $5(c)$ during an optimal flooding	64
5.1	Gadget G_{x_i}	76
5.2	Gadget G_{c_j} .	77
5.3	Graph obtained from $F = (x_1)(x_2)(x_1 + \bar{x_2} + x_3)(\bar{x_1} + \bar{x_2} + \bar{x_3})(\bar{x_3})$	78
5.4	Gadget G_{x_i} and its subgraph B_{x_i} inside the rectangle. The white vertices	
	are pendant vertices in G and are not contained in B_{x_i}	79
5.5	Six internal vertices to contaminate a gadget g_{u_i}	81
5.6	Two vertices in $\{u_j, u_{j+1}, u_{j+2}, u_{j+3}\}$ are contaminated implying to use only	
	five internal vertices to contaminate the gadget g_{u_i}	81
5.7	$(a) - (d)$ Choices of vertices in S_A that imply in at least 5 vertices to be	
	added to S_A ; thicker edges mean that one of its endpoints must be added	
	to S_A ; $(e) - (h)$ Choices of vertices in S_A that imply in exactly 4 vertices	
	to be added to S_A	83
5.8	(a) Satisfiable boolean formula $F = (x_1)(x_2)(x_1 + \neg x_2)(\neg x_1 + \neg x_2 + \neg x_2)(\neg x_1 + \neg x_2)(\neg x_1)(\neg x_1)(\neg$	
	$\neg x_3$)($\neg x_3$); (b) Graph G constructed from F	84

Chapter 1 Introduction

The question "P = NP?" is the most important open question in computer science, and the theory of NP-completeness was developed to show which problems probably do not have polynomial-time algorithms. However, many NP-hard and NP-complete problems must still be solved in practice; therefore it is natural to ask if each of these problems admits an algorithm whose non-polynomial time complexity is purely a function of some subset of its aspects. Questions about the existence of such algorithms are addressed within the theory of parameterized computational complexity developed by Downey and Fellows.

In this thesis we develop a systematic complexity analysis of these problems, defining its subproblems and mapping which one belongs to each side of an "imaginary boundary" between polynomial-time solvability and intractability. After that, we analyze which sets of aspects of these problems are *sources* of their intractability, that is, subsets of aspects for which there exists an algorithm to solve the associated problem, whose nonpolynomial time complexity is purely a function of these sets. Thus, we use classical and parameterized computational complexity in an alternate and complementary approach, to show which subproblems of the given problems are NP-hard and latter to diagnose for which sets of parameters the problems are fixed-parameter tractable, or in FPT.

This thesis exhibits a classical and parameterized complexity analysis of different groups of NP-hard problems. The addressed problems are divided into four groups of distinct nature, in the context of data structures, combinatorial games, and graph theory: (I) and/or graph solution and its variants; (II) flooding-filling games; (III) problems on P_3 -convexity; (IV) problems on induced matchings.

The first group of problems involve two important data structures used for modeling many real-word applications, and/or graphs and x-y graphs. An and/or graph is an acyclic digraph containing a source, such that every vertex $v \in V(G)$ has a label $f(v) \in \{\text{and,or}\}$. X-y graphs are a generalization of and/or graphs: every vertex v_i of an x-y graph has a label x_i - y_i meaning that v_i depends on x_i of its y_i out-neighbors. We investigate the complexity of finding a solution subgraph H of such digraphs, which must contain the source and obey the following rule: if a vertex is included in H then x_i of its out-edges must also be included in H, where an and-vertex has $x_i = y_i$, and an or-vertex has $x_i = 1$.

The second group of problems consists of variants of a one-player combinatorial game known as the Flood-Filling Game, which is played on a colored board and whose objective is to make the board monochromatic ("flood the board") with the minimum number of flood moves. A flood move consists of assigning a new color c_i to the a pivot tile p and also to all the tiles connected to p by a monochromatic path immediately before the move. The flood-filling game where all moves use the same pivot p is denoted by Flood-It. When the player can freely choose which tile will be the pivot of each move the game is denoted by Free-Flood-It.

The third group comprises some problems on P_3 -convexity. More specifically we are interested in identifying either the minimum P_3 -geodetic set or the minimum P_3 -hull set S of a graph, from which the whole vertex set of G is obtained either after one or eventual iterations, respectively. Each iteration adds to a set S' of vertices all the vertices of $V(G) \setminus S'$ with two neighbors in S'.

The last group of problems studied in this thesis focus on a classical topic in graph theory. These problems are related to maximum matchings, maximum induced matchings, and the distance between them in a graph. The matching number $\nu(G)$ of G is the maximum cardinality of a matching in G, and a matching with $\nu(G)$ edges is a maximum matching of G. An induced matching is a set M' of edges of G at pairwise distance at least 2. The induced matching number $\nu_2(G)$ of G is the maximum cardinality of an induced matching in G, and an induced matching with $\nu_2(G)$ edges is a maximum induced matching. The distance between a maximum matching of a graph G and its maximum induced matching is the difference between the cardinality of these sets ($\nu(G) - \nu_2(G)$).

For each group of problems above, there is a chapter in this thesis devoted to it. A brief abstract of our work and the obtained results will be presented at the beginning of each chapter.

1.1 Background

A computational problem is a question to be answered, typically containing several variables whose values are unspecified. An *instance* of a problem is created by specifying particular values for its variables. A problem is described by specifying both its instances and the nature of solutions for those instances.

A decision problem Π consists of a set D_{Π} of instances and a set $Y_{\Pi} \subseteq D_{\Pi}$ of yesinstances. A decision problem is described informally by specifying: (i) a generic instance in terms of its variables; (ii) a yes-no question stated in terms of the generic instance. An optimization problem Π consists of a set D_{Π} of instances and a set $S_{\Pi} \subseteq D_{\Pi}$ of solutions such that for each $I \in D_{\Pi}$, there is an associated set $S_{\Pi}[I] \subseteq S_{\Pi}$ of solutions for I. An optimization problem is described informally by specifying: (i) a generic instance in terms of its variables; (ii) an objective function g to be calculated, and the properties that must be satisfied by any solution associated with an instance created from the generic instance. An optimal solution $S_{\Pi}[I]$ is a solution which maximizes/minimizes the value $g(S_{\Pi}[I])$.

An algorithm A for a problem Π is a finite sequence of instructions for some computer which solves Π . A polynomial time algorithm is defined to be one whose time complexity function is O(p(n)) for some polynomial function p, where n is used to denote the input length [41].

Definition 1.1.1. A problem Π belongs to class P if and only if Π can be solved in polynomial time by a deterministic algorithm.

Definition 1.1.2. A problem Π belongs to class NP if and only if for a given certificate (a string that certifies the answer to a computation), there is a deterministic algorithm which verifies its validity in polynomial time.

Definition 1.1.3. Given two problems Π and Π' , $\Pi \propto \Pi'$ (Π is reducible to Π' in polynomial time) if there exists an algorithm that, given an instance I of Π , constructs an instance I' of Π' in polynomial time in |I| such that from a subroutine to I', a correct answer for I is output.

Definition 1.1.4. A problem Π' is NP-hard if for all problems $\Pi \in NP$, $\Pi \propto \Pi'$; if Π' is also in NP, then Π' is NP-complete.

It is easy to see that $\Pi \in P$ implies $\Pi \in NP$. If any single NP-hard problem can be solved in polynomial time, then all problems in NP can also be solved in polynomial time. If any problem in NP cannot be solved in polynomial time, then so neither can all NPcomplete problems. An NP-complete problem Π , therefore, has the following property: $\Pi \in P$ if and only if P = NP. The question "P = NP?" is the most important open question in computer science.

An algorithm is *efficient* if its complexity function satisfies some criterion, e.g., the complexity function is a polynomial in the instance size. A problem is *tractable* if it has an efficient algorithm; otherwise, the problem is said to be *intractable*. As there are many possible criteria which can be used to define efficiency, there are many possible types of tractability and intractability [91].

1.1.1 Parameterized Tractability in Pratice

The theory of NP-completeness was developed to show which problems do not probably have polynomial time algorithms. Since the beginning of this theory in 1971, thousands of other problems have been shown to be NP-hard and NP-complete. Though it is nice to know that such problems do not have polynomial time algorithms unless P = NP, a inconvenient fact remains: these problems (especially those with real-world applications) must still be solved. Thus, the following question emerges:

"How does one solve NP-complete problems efficiently in practice?"

Firstly, we have two possibilities:

- × Try to construct a polynomial time algorithm (implies P = NP).
- $\sqrt{}$ Invoke some type of polynomial-time heuristic algorithm.

However, in pratice some set of aspects of the problem has bounded size or value. There are another approaches:

- × Invoke some type of "brute force" optimal-solution technique, that in effect runs in polynomial time because its time complexity function is $O(n^{f(k)})$, where n is used to denote the input length and k is some aspect with bounded size or value. However, when the instances to be solved are large, this approach may not be feasible.
- Invoke a non-polynomial time algorithm such that its non-polynomial time complexity is *purely* a function of some subset of aspects of the problem that are of bounded size or value in practice.

This last approach immediately suggests the following questions:

- 1. Given a problem and a subset of aspects of that problem, is there an algorithm for that problem whose non-polynomial time complexity is purely a function of those aspects?
- 2. Relatively to which aspects of that problem do such algorithms exist?

If a problem Π for a set K of its aspects admits such algorithms described in (1), i.e. solvable in $f(K).n^{O(1)}$ time, then $\Pi \in FPT$ with respect to K (the class of *fixed-parameter tractable* problems). Alternatively, one can show that such algorithm probably does not exist by establishing a intractability of this version of the problem.

1.1.2 Multivariate Investigation

According to Garey and Johnson [41], whenever we are confronted with a new problem, a natural first question to ask is: Can it be solved via a polynomial time algorithm? We can concentrate our efforts on trying to find a polynomial time algorithm as efficient as possible. However, if no polynomial time algorithm is apparent, an appropriate second question to ask is: "Is the problem NP-complete?". Suppose now we have just succeeded in demonstrating that our initial problem is NP-complete. Even though this answers the two questions which began our analysis, there are still many appropriate follow-up questions that could be asked. The problem we have been analyzing is often distilled from a less elegant applied problem, and some of the details that were dropped in the distillation process might alter the problem enough to make it polynomially solvable. If not, there still might be significant special cases that can be solved in polynomial time. Such possibilities can be investigated by analyzing subproblems of our original problem.

It should be apparent that, even though a problem Π is NP-complete, each of the subproblems of Π might independently be either NP-complete or polynomially solvable. Assuming that $P \neq NP$, we can view the subproblems of any NP-complete problem Π as lying on different sides of an imaginary "boundary" between polynomial time solvability and intractability [41].

Figure 1.1 [41] gives a schematic representation for one possible "current state of knowledge" about a collection of subproblems of a problem Π .



Figure 1.1: One possible state of knowledge about subproblems of an NP-complete problem Π . An arrow from Π^1 to Π^2 signifies that Π^1 is a subproblem of Π^2 .

In this thesis, our first goal when analyzing a problem is to determine which subproblems lie on each side.

Intractability Mapping

Any problem Π contains a domain D that is the set of all instances of Π . A problem Π' is a subproblem of Π if it asks the same question as Π , but only over a subset of the domain of Π .

Definition 1.1.5. Let Π be a problem with domain D and let $C = \{a_1, a_2, \ldots, a_\ell\}$ be a subset of aspects of Π . We denote by:

- $[a_1=c'_1,a_2=c'_2,\ldots,a_\ell=c'_\ell]$ - Π the subproblem of Π with domain D' such that each instance in D' has aspects a_1, a_2, \ldots, a_ℓ bounded by the constants $c'_1, c'_2, \ldots, c'_\ell$ respectively.
- [a₁,a₂,...,a_ℓ]-Π, or [C]-Π, the family of variants of Π such that every aspect in C is bounded by some constant.

Given an NP-hard problem Π and a subset C of its aspects, a systematic complexity analysis starts from the following steps.

- 1. Verify if [C]- Π is in P, or NP-hard.
- 2. If [C]- Π is in P, determine each minimal subset C' of C such that [C']- Π is in P.
- 3. If [C]- Π is NP-hard, determine for which values of the aspects in C the problem is solvable in polynomial time or remains NP-hard.

In a systematic complexity analysis of a problem Π , it is very common to identify subproblems of Π which can be solved in polynomial time. In general, it can be shown by some exhaustive algorithm in time $O(n^{f(k)})$, where n is used to denote the input length and k is some aspect with bounded size or value. Note that, when the instances to be solved are large, this approach may not be feasible in practice.

A parameter is a function which extracts a particular aspect or set of aspects of a problem from instances of that problem; it can also be considered as that set of aspects. As such, a parameter is both a mechanism for isolating an aspect of a problem and the "container" in which these aspects are packaged for subsequent manipulation [91].

Definition 1.1.6. A parameterized problem Π is described informally by specifying:

- A generic instance in terms of its variables.
- The aspects of an instance that comprise the parameter.
- A question stated in terms of the generic instance.

Definition 1.1.7. Let Π be a NP-hard problem and let $S = \{a_1, a_2, \ldots, a_\ell\}$ be a subset of aspects of Π . We denote by:

• $\Pi(a_1, a_2, \ldots, a_\ell)$, or $\Pi(S)$, the parameterized version of Π where the aspects in S are fixed as parameters.

Definition 1.1.8. [31] A parameterized problem $\Pi(S)$ belongs to the class XP if there exists an algorithm to solve $\Pi(S)$ in time $f(S).n^{g(S)}$, where n is used to denote the input length and f and g are arbitrary functions.

Observation 1.1.1.

 [S]-Π and Π(S) are different problems. The instances of [S]-Π has the aspects in S with size bounded by constants, while in Π(S) the parameters are just a mechanism for isolating aspects for subsequent manipulation (in this case, the aspects not necessarily have bounded size).

Lemma 1.1.1. Given an NP-hard problem Π and a subset S of its aspects, if [S]- Π remains NP-hard, then the parameterized problem $\Pi(S)$ is not in XP, unless P = NP.

Proof. If Π is in XP then by definition this problem is solved by an algorithm that runs in time $f(S)n^{g(S)}$ for some functions f and g. When the value of every aspect in S is fixed, the values of f(S) and g(S) become constants and this running time becomes polynomial in n. As this algorithm also solves [S]- Π and [S]- Π is NP-hard then P = NP.

Corollary 1.1.1. If $P \neq NP$, then $\Pi(S)$ is in XP if and only if [S]- Π is solvable in polynomial time.

Given a problem Π and some subset $S = \{s_1, \ldots, s_n\}$ of the aspects of Π , there are 3^n different basic families of variants of the problem, based on which of the aspects is declared as either:

- 1. an unrestricted part of the input,
- 2. part of the aggregate parameterization, or
- 3. a fixed constant (yielding part of the indexing of the family of parameterized problems).

Let Π be problem and let $S = \{s_1, \ldots, s_n\}$ be a subset of the aspects of Π . $[S_1]$ - $\Pi(S_2)$ is the family of parameterized problems where the aspects in $S_1 \subseteq S$ are fixed constants and the aspects in $S_2 \subseteq S$ are aggregate parameters.

A parameterized problem $\Pi(S)$ belongs to the class FPT, or fixed-parameter tractable, if there exists an algorithm to solve $\Pi(S)$ in time $f(S).n^{O(1)}$, where n is used to denote the input length and f is an arbitrary function.

Individual parameterized results are very good at establishing whether or not a given problem has an FPT-algorithm for a particular set of aspects of that problem. However, if one is interested in fully characterizing the set of FPT-algorithms for parameterized versions of a NP-hard problem, individual results are not sufficient because a fixed-parameter tractability (intractability) result says nothing about which subsets (supersets) of its aspects also render fixed-parameter tractability (intractability) [91]. In this case, it is necessary to make a systematic parameterized complexity analysis of the problem, determining the parameterized complexity relative to all non-empty subset of aspects of the problem.

A list of parameterized results produced by a systematic parameterized complexity analysis relative to some set of aspects S for a problem Π can be visualized as a *polynomial time intractability map* that shows which sets of aspects of the problem can be said to be responsible for (and hence are sources of) that problem's polynomial time intractability [91].

Definition 1.1.9. [91] Given a NP-hard problem Π and some subset S of aspects of Π , S is a source of polynomial-time intractability for Π , if $\Pi(S)$ is in FPT.

" In parameterized complexity, the focus is not on *whether* a problem is hard, the theory starts from the assumption that most interesting problems are intractable when considered classically. The focus is on the question: *What makes the problem computationally difficult?* ". [31]

In this thesis, one of the goals it is to make an analysis on the sources of polynomial time intractability of some problems, that are "minimal" in the sense that their associated FPT-algorithms are not trivial extensions of other FPT-algorithms.

1.2 Organization of this Thesis

The remainder of this work is organized as follows.

In Chapter 2, we present a more detailed review of the theory of parameterized complexity, where the concepts and some techniques of fixed-parameter tractability and intractability will be discussed. We also present the concept of polynomial kernel.

Chapter 3 presents a study on two data structures that have been used to model several problems in computer science: and/or graphs and x-y graphs. We analyze the classical and parameterized complexity of the optimization problems MIN-AND/OR and MIN-X-Y, which consist of finding solution subgraphs of optimal weight for and/or and x-y graphs, respectively. The first results presented in this chapter have been published in the paper [88]:

U. S. Souza, F. Protti and M. Dantas da Silva, "*Revisiting the Complexity of* And/Or Graph Solution", Journal of Computer and System Sciences, 79-7, 2013, 1156-1163.

We prove that:

- 1. MIN-AND/OR remains NP-hard even for a very restricted family of and/or graphs where edges have weight one and or-vertices have out-degree at most two;
- 2. deciding whether there is a solution subtree with weight exactly k of a given x-y tree is also NP-hard;
- 3. the parameterized problem MIN-AND/OR⁰(k), whose domain includes and/or graphs allowing zero-weight edges, is W[2]-hard;
- 4. the parameterized problem MIN-X-Y(k) is W[1]-hard.

In the end of Chapter 3, we close the main open question which still remained open in [88]: "Is the problem of finding a solution subgraph of cost at most k (where k is a fixed parameter) in FPT?". We answer affirmatively to this question, via kernelization techniques. Also, using a framework developed by Bodlaender *et al.* (2009) and Fortnow and Santhanam (2011), based upon the notion of compositionality, we show that the above parameterized problem does not admit a polynomial kernel unless $NP \subseteq coNP/poly$.

These last results have been previously presented in the extended abstract [86]:

U. S. Souza, F. Protti and M. Dantas da Silva, "*Parameterized And/Or Graph Solution*", **12th Cologne Twente Workshop on Graphs and Combinatorial Optimization, CTW 2013**, 205-208.

In addition, during our research, we have also published the following works on applications of and/or graphs [84, 66]:

R. P. Medeiros, U. S. Souza, F. Protti, L. G. P. Murta, "Optimal Variability Selection in Product Line Engineering", Proc. of the 24th International Conference on Software Engineering and Knowledge Engineering - SEKE 2012, 635-640.

U. S. Souza, F. Protti, M. Dantas da Silva, "Complexidade Parametrizada para Problemas em Grafos E/OU", Pesquisa Operacional para o Desenvolvimento, 4-2, 2012, 160-174.

Chapter 4 presents new results on flood-filling games, Flood-It and Free-Flood-It, in which the player aims to make the board monochromatic with a minimum number of flooding moves.

First, a complete mapping of the complexity of flood-filling games on trees is made, charting the consequences of single and aggregate parameterizations by: number of colors, number of moves, maximum distance of the pivot, maximum orbit, number of leaves, and number of "bad moves".

We present some polynomial time and parameterized tractability and intractability results. We also show that Flood-It on trees and Restricted Shortest Common Supersequence (RSCS) are analogous problems, in the sense that they can be translated from one to another, keeping complexity features; this implies that Flood-It on trees inherits several complexity results already proved for RSCS, such as some interesting FPT and W[1]-hard cases. In addition, we prove that Flood-It remains NP-hard when played on 3-colored trees, which closes an open question. We also present a general framework for reducibility from Flood-It to Free-Flood-It; some NP-hard cases for Free-Flood-It on trees can be derived using this approach.

These results have been published in the paper [87]:

U. S. Souza, F. Protti and M. Dantas da Silva, "Parameterized Complexity of Flood-Filling Games on Trees", 19th International Computing & Combinatorics Conference, COCOON 2013, LNCS 7936, 531-542.

We also analyze the behavior of these games when played on other classes of boards, such as powers of cycles, circular grids and graphs with bounded vertex cover. We describe polynomial time algorithms to play Flood-it on C_n^2 (the second power of a cycle on n vertices), $2 \times n$ circular grids, and some types of d-boards (grids with a monochromatic column). We also show that Free-Flood-it is NP-hard on C_n^2 and $2 \times n$ circular grids.

These last results have been presented in [85]:

U. S. Souza, F. Protti and M. Dantas da Silva, *"Inundação em Grafos"*, 16th Congreso Latino Iberoamericano de Investigación Operativa & 44th Simpósio Brasileiro de Pesquisa Operacional, CLAIO/SBPO 2012.

Moreover, in Chapter 4 we also study the parameterized complexity of such problem considering the size of the minimum vertex cover as parameter. We show that Flood-it is fixed-parameter tractable when parameterized by the size of the minimum vertex cover, and admits a polynomial kernelization when considering the number of colors as a second parameter.

In the following chapters, I describe the works done in cooperation with professors Dieter Rautenbach and Lucia Draque Penso during the doctoral internship at the University of Ulm, Germany, in the period November, 2013 to February, 2014.

In Chapter 5 we study new complexity aspects of P_3 -convexity restricted to graphs with bounded maximum degree. More specifically we are interested in identifying either the minimum P_3 -geodetic set or the minimum P_3 -hull set of such graphs. We prove that:

- 1. the minimum P_3 -hull set of a graph G can be found in polynomial time when $\delta(G) \geq \frac{n(G)}{c}$ (for some constant c);
- 2. determining the size of the minimum P_3 -hull set of a graph remains NP-hard even on planar graphs with maximum degree four;
- 3. the minimum P_3 -hull set of a cubic graph can be found in polynomial time;
- 4. determining the size of the minimum P_3 -hull set of a graph remains NP-hard even on planar graphs with maximum degree three;
- 5. the minimum P_3 -hull set can be found in polynomial time in graphs with minimum feedback vertex set of bounded size and no vertex of degree two;
- 6. it is NP-hard to determine the size of the minimum P_3 -geodetic set of a planar graph with maximum degree three.

The results comprised in this chapter are contained in the following paper [76]:

L. D. Penso, F. Protti, D. Rautenbach, U. S. Souza, "On P_3 -convexity of Graphs with Bounded Degree", 10th International Conference on Algorithmic Aspects of Information and Management, AAIM 2014.

In Chapter 6 we study graphs G with $\nu(G) - \nu_2(G) \leq k$, where $\nu(G)$ is the cardinality of the maximum matching of G, and $\nu_2(G)$ is the cardinality of its maximum induced matching. We show that the recognition of these graphs can be done in polynomial time for fixed k, and it is fixed parameter tractable when parameterized by k for graphs of bounded maximum degree. Finally, we extend some of Cameron and Walker's results to k-matchings in graphs of sufficiently large girth.

Chapter 2

Parameterized Complexity

"Half of science is asking the right questions."

Roger Bacon

Classical complexity views a problem as an instance and a question, where the running time is specified by the input's size. However, when a problem comes from "real life" we always know more about the problem. The problem is planar, the problem has small width, the problem only concerns small values of the parameters. Thus, why not have a complexity theory which exploits these structural parameters? Why not have a complexity theory more fine-tuned to actual applications? [31]

The Parameterized Complexity Theory was proposed by Downey and Fellows [31] as a promising alternative to deal with NP-hard problems described by the following general form [75]: given an object x and a nonnegative integer k, does x have some property that depends only on k (and not on the size of x)? In parameterized complexity theory, k is set as the *parameter*, considered to be *small* in comparison with the size |x| of object x. It may be of high interest for some problems to ask whether they admit deterministic algorithms whose running times are exponential with respect to k but polynomial with respect to |x|.

As is common in complexity theory, we describe problems as languages over finite alphabets Σ . To distinguish them from parameterized problems, we refer to sets $\Pi \subseteq \Sigma^*$ of strings over Σ (nonempty) as classical problems [37].

Definition 2.0.1. [37] Let Σ be a finite alphabet.

1. A parametrization of Σ^* is a mapping $k : \Sigma^* \to \mathbb{N}$ that is polynomial time computable. 2. A parameterized problem (over Σ) is a pair $(\Pi, k) = \Pi(k)$, consisting of a set $\Pi \subseteq \Sigma^*$ and a parametrization k of Σ^* .

Example. Let SAT denote the set of all satisfiable propositional formulas, where propositional formulas are encoded as strings over some finite alphabet Σ . Let $k : \Sigma^* \to \mathbb{N}$ be the parameterization defined by:

$$k = \begin{cases} number of variables of x, & if x is a formula with at least one variable, \\ 1, & otherwise. \end{cases}$$

If $\Pi(k)$ is a parameterized problem over the alphabet Σ , then we call strings $x \in \Sigma^*$ instances of $\Pi(k)$ and k, the the corresponding parameter. Usually, we represent a parameterized problem $\Pi(k)$ in the form:

Instance: $x \in \Sigma$. Parameter: k Problem: Decide whether $x \in \Pi(k)$.

In the same way that the notion of *polynomial time* is central to the classical formulation of computational complexity, a central notion to parameterized complexity is *fixed-parameter tractability*.

Definition 2.0.2. [37] A parameterized problem $\Pi(k)$ is fixed-parameter tractable, or FPT, if the question " $x \in \Pi(k)$?" can be decided in running time $f(k).|x|^{O(1)}$, where fis an arbitrary function on nonnegative integers. The corresponding complexity class is called FPT.

2.1 Bounded Search Tree Technique

According to Downey and Fellows [31], the method of bounded search trees is probably the easiest to apply and is based on the following facts. Many combinatorial problems can be solved by algorithms that can be decomposed into two distinct parts:

- First, within the algorithm we compute, perhaps inefficiently, some search space which is often an exponential-size search tree.
- Thereafter, we run some relatively efficient algorithm on each branch of the tree simply based upon, say, depth first search.

The exponential worst-case complexity of such algorithms comes from problem instances where we need complete tree traversal. For our purposes, the critical observation

(2.1)

is that for many parameterized problems, the size of the tree depends only upon the parameter. Then, for a fixed k, the search space becomes constant in size and the algorithm is then efficient for each fixed k [31].

A vertex cover of a graph is a set of vertices such that each edge of the graph is incident to at least one vertex of the set. The problem of finding a minimum vertex cover is a classical optimization problem in computer science. Its decision version, VERTEX COVER, was one of Karp's 21 NP-complete problems.

Example. [31] VERTEX COVER is solvable in time $O(2^k . n^{O(1)})$, where it is asked if a graph G has a vertex cover of size k.

Proof. We construct a binary tree of height k as follows. Label the root of the tree with the empty set and the graph G. Choose an edge $(u, v) \in E$. In any vertex cover VC of G, we must have either $u \in VC$ or $v \in VC$, so we create children of the root node corresponding to these two possibilities. Thus, the first child is labeled with $\{u\}$ and G - u, and the second child is labeled with $\{v\}$ and G - v. The set of vertices labeling a node represents a "possible" vertex cover, and the graph labeling the node represents what remains to be covered in G. In general, for a node labeled with the set of vertices S and the subgraph H of G, we choose an edge $(u, v) \in E(H)$ and create two child nodes labeled, respectively, $S \cup \{u\}$ and H - u, and $S \cup \{v\}$ and H - v. If we create a node at height at most k in the tree that is labeled with a graph having no edges, then a vertex cover of cardinality at most k has been found. There is no need to explore the tree beyond height k.

2.2 Kernelization

According to [11], a fundamental and very powerful technique in designing FPT algorithms is *kernelization*. In a nutshell, a kernelization algorithm for a parameterized problem is a polynomial-time transformation that transforms any given instance to an equivalent instance of the same problem with size and parameter bounded by a function of the parameter in the input. Typically this is done using so-called *reduction rules*, which allow the safe reduction of the instance to an equivalent "smaller" instance. In this sense, kernelization can be viewed as polynomial-time preprocessing which has universal applicability, not only in the design of efficient FPT algorithms, but also in the design of approximation and heuristic algorithms [46].

Definition 2.2.1. [37] Let $\Pi = (I, k)$ be a parameterized problem, where instance I is asked to have a solution of size k. Reduction to problem kernel means to replace instance (I, k) by a reduced instance (I', k') (called problem kernel, or just kernel) such that $k' \leq ck$ for a constant c, $|I'| \leq g(k)$ for some function g only depending on k, and $(I, k) \in \Pi$ if and only if $(I', k') \in \Pi$. Furthermore, the reduction from (I, k) to (I', k') is computable in polynomial time.

Example. [31] VERTEX COVER admits a kernel of size $O(k^2)$, where k is the parameter corresponding to the size of a vertex cover.

Proof. Observe that for a simple graph G any vertex of degree greater than k must belong to every k-element vertex cover of G.

Step 1: Locate all vertices in G of degree greater than k; let p be the number of such vertices. If p > k, there is no k-vertex cover. Otherwise, let k' = k - p.

Step 2: Discard all p vertices found in step 1 and the edges incident with them. If the resulting graph G' has more than k'(k+1) vertices, reject.

Step 3: If G' has no k'-vertex cover, reject. Otherwise, any k'-vertex cover of G' plus the p vertices from step 1 constitutes a k-vertex cover of H.

The bound k'(k+1) in step 2 is jutisfied by the fact that a simple graph with a k'-vertex cover and degree bounded by k has no more than k'(k+1) vertices. As we can see in step 3, G' is a kernel for the problem.

Lemma 2.2.1. [9] Let Π be a parameterized problem. Then Π belongs to the class FPT if and only if Π is decidable and Π has a kernel.

There are also generalizations of the above definition that have appeared in the literature. Most notably is the generalization which allows the kernelization algorithm to map the input instance to an instance of a different language [11].

Definition 2.2.2. [11] A generalized kernelization algorithm from a parameterized problem $\Pi(k)$ to another parameterized problem $\Pi'(k')$, where $k' \leq f(k)$, is an algorithm that given an instance x of $\Pi'(k')$, outputs in polynomial time on (|x| + k) an instance x' of $\Pi'(k')$ such that:

- 1. $x \in \Pi(k)$ if and only if $x' \in \Pi'(k')$,
- 2. $|x'| \le f(k)$.

While the latter definition can prove useful in certain cases, where one can, for instance, generalize the problem at hand in order to obtain additional combinatorial leverage, the former definition is the more natural one [11].

2.3 Parameterized Intractability

To establish that a parameterized problem is fixed-parameter intractable we need the following additional definitions.

Definition 2.3.1. [37] Let $\Pi(k)$ and $\Pi'(k')$ be parameterized problems over alphabets Σ and Σ' , respectively, where $k' \leq g(k)$ for some computable function $g : \mathbb{N} \to \mathbb{N}$. An FPTreduction (or parametric reduction) from $\Pi(k)$ to $\Pi(k')$ is a mapping $R : \Sigma^* \to (\Sigma')^*$ such that:

- 1. For all $x \in \Sigma^*$, it holds that $x \in \Pi(k)$ if and only if $R(x) \in \Pi'(k')$;
- 2. R is computable by an FPT-algorithm (with respect to k);

If a parametric reduction exists between Π and Π' then Π parametrically reduces to Π' .

Lemma 2.3.1. (transitivity) Given parameterized problems Π , Π' and Π'' , if Π parametrically reduces to Π' and Π' parametrically reduces to Π'' then Π parametrically reduces to Π'' .

Lemma 2.3.2. (preservation of fixed-parameter tractability) Given parameterized problems Π and Π' , if Π parametrically reduces to Π' and Π' is fixed-parameter tractable then Π is fixed-parameter tractable.

The following definitions give the parameterized analogs of the class NP in the theory of NP-completeness. These classes are, for the most part, based on a series of successively more powerful solution-checking circuits in which solutions are encoded as input vectors to these circuits and parameters are encoded in the weights of these input vectors [91].

Definition 2.3.2. [31] Let C be a decision circuit with input variables x_1, \ldots, x_n . The weft of C is defined to be the maximum number of large gates on any path from the input variables to the output line. (A gate is called large if its fan-in exceeds some preagreed bound, in general two)

Definition 2.3.3. The weight of an assignment to the variables of a boolean circuit C (for short, an assignment to C) is the number of 1's in that assignment.

Weighted Weft t Depth h Circuit Satisfiability - WCS(t,h)Instance: A weft t depth h decision circuit C. Parameter: A positive integer k. Question: Does C have a satisfying assignment with weight k?

Definition 2.3.4. [31] We define a parameterized problem Π to be in the class W[t] if and only if Π is FPT-reducible to WCS(t,h).

Definition 2.3.5. (The W-hierarchy) We term the union of these W[t] classes together with three other classes $W[SAT] \subseteq W[P] \subseteq XP$, the W-hierarchy. Here, W[P] denotes the class obtained by having no restriction on depth, i.e., P-size circuits, and W[SAT]denotes the restriction to boolean circuits of P-size. Hence, the W-hierarchy is

$$FPT \subseteq W[1] \subseteq W[2] \subseteq \ldots \subseteq W[SAT] \subseteq W[P] \subseteq XP.$$

Downey and Fellows have conjectured that each of the containments in the Whierarchy is proper [31].

Observation 2.3.1. To prove that a parameterized problem Π belongs to a class W[t], $t \geq 1$, it is sufficient to show a FPT-reduction to some parameterized problem in W[t]. See Lemma 2.3.1.

In addition to the W-hierarchy, in parameterized complexity there are another hierarchies of classes of parameterized problems, such as the M-hierarchy and A-hierarchy. However, the results presented in this thesis focus only on the W-hierarchy.

In parameterized complexity, we define Q-hardness and Q-completeness of a parameterized problem $\Pi(k)$ with respect to a complexity class Q, as in classical complexity theory: $\Pi(k)$ is Q-hard under FPT-reductions if every problem in Q is FPT-reducible to $\Pi(k)$; $\Pi(k)$ is Q-complete under FTP-reductions if $\Pi(k) \in Q$ and $\Pi(k)$ is Q-hard.

2.3.1 Analog of Cook's Theorem

Downey and Fellows [31] proved an analog of Cook's Theorem by proving that a number of combinatorial problems are of the same fixed-parameter complexity as a generic problem about nondeterministic Turing machines. The key underlying problems are the following.

Short Turing Machine Acceptance

Instance: A nondeterministic Turing machine \mathbb{M} and a string x.

Parameter: A positive integer k.

Question: Does \mathbb{M} have a computation path accepting x in at most k steps?

Weighted q-CNF Satisfiability

Instance: A boolean expression F in conjunctive normal form (CNF) such that each clause has no more than q literals ($q \ge 2$).

Parameter: A positive integer k.

Question: Does F have a satisfying truth assignment of weight k?

Theorem 2.3.1. [31] (Analog of Cook's Theorem) The following are complete for W[1]:

1. Short Turing Machine Acceptance.

2. Weighted q-CNF Satisfiability.

From Theorem 2.3.1 many other parameterized problems have been proved to be W[1]complete, such as Clique(k) and Independet Set(k), where k is the parameter for the size of the subsets.

The Analog of Cook's Theorem can be used as a tool to show W[1]-hardness and W[1]completeness of parameterized problems. To establish some results about W[t] classes for $t \ge 2$, Downey and Fellows presented a higher-level version of Cook's theorem, the *Normalization Theorem*.

Definition 2.3.6. We say that a propositional formula F is t-normalized if F is of the form products-of-sums-of-products... of literals with t-alternations.

Note that a formula 2-normalized is the same as CNF.

Weighted *t*-normalized Satisfiability

Instance: A t-normalized boolean expression $F(t \ge 2)$.

Parameter: A positive integer k.

Question: Does F have a satisfying truth assignment of weight no more than k?

Theorem 2.3.2. [31] (The Normalization Theorem) Weighted t-normalized Satisfiability is complete for W[t], for all $t \ge 2$.

Using this last theorem many other parameterized problems were shown to be W[t]hard or W[t]-complete, for some $t \ge 2$. As example, the parameterized problem DOM-INATING SET(k) (k is the size of the dominating set) was shown to be W[2]-complete. Several other results can be found in [31].

2.4 Infeasibility of Polynomial Kernels

Since every FPT problem has a kernelization algorithm, it is interesting to study problems which allow kernelization algorithms that reduce instances to a size which is polynomially bounded by the parameter. Such problems are said to have a *polynomial kernelization algorithm*, or a *polynomial kernel*. While positive kernelization results have appeared regularly over the last two decades, the first results establishing infeasibility of polynomial kernels for specific problems have appeared only recently. In particular, Bodlaender et al. [11] and Fortnow and Santhanam [38] have developed a framework based upon the notion of *compositionality*, for showing that a problem does not admit a polynomial kernel unless $NP \subseteq coNP/poly$, implying a collapse of the polynomial hierarchy to the third level $(PH = \Sigma_p^3)$, which is deemed unlikely.

Fortnow and Santhanam [38] first showed the infeasibility of compressing what they called *OR-SAT*. Let ϕ_i , $1 \leq i \leq t$, be instances of *SAT*, which is the standard language of

propositional Boolean formulas. OR-SAT asks if there is a function f that, given as input m boolean formulas ϕ_1, \ldots, ϕ_m where each ϕ_i has length at most n, outputs a boolean formula, and has the following properties:

- f is computable in time polynomial in m and n,
- $f(\phi_1, \ldots, \phi_m)$ is satisfiable if and only if at least one of the ϕ_i is satisfiable,
- $|f(\phi_1, \ldots, \phi_m)|$ is bounded by a polynomial in n.

As in [70], any compression routine for OR-SAT can be thought of as an algorithm that accepts multiple instances of SAT and returns a small formula equivalent to the "or" of the input formulas. A *Or-distillation algorithm* for a given problem is designed to act as a "Boolean OR of problem-instances", it receives as input a sequence of instances, and produces a YES-instance if and only if at least one of the instances in the sequences is also a YES-instance. Although the algorithm is allowed to run in time polynomial in the total length of the sequence, its output is required to be an instance whose size is polynomially bounded by the size of the maximum-size instance in its input sequence. Formally, we have the following:

Definition 2.4.1. (Or-distillation [11]) Let Π be an NP-complete problem. An Ordistillation of Π is a polynomial time algorithm D that receives as input a series of m instances of Π , and outputs one instance of Π , such that

- If D has as input m instances, each of size at most n, then D uses time polynomial in m and n, and its output is bounded by a function that is polynomial in n.
- If D has as input instances x_1, \ldots, x_m , then $D(x_1, \ldots, x_m) \in \Pi$ if and only if $\exists_{1 \leq i \leq m} x_i \in \Pi$.

Theorem 2.4.1. ([11]) If any NP-complete problem has an Or-distillation algorithm then $NP \subseteq coNP/poly$.

Combined with Yap's theorem [92] $(NP \subseteq coNP/poly \Rightarrow PH \subseteq \Sigma_3^p)$, this will also imply that a distillation algorithm for an NP-complete problem implies a collapse of the polynomial hierarchy to the third level.

Definition 2.4.2. (Or-composition [11]) Let $\Pi \subseteq L^* \times N$ be a parameterized problem. An Or-composition of Π is a polynomial time algorithm D that receives as input a sequence $((x_1, k), (x_2, k), \dots, (x_m, k))$, with each $(x_i, k) \in L^* \times N$, and outputs a pair (x', k'), such that

- the algorithm uses time polynomial in $\sum_{1 \le i \le m} |x_i| + k$;
- k' is bounded by a polynomial in k;

• $(x', k') \in \Pi$ if and only if $\exists_{1 \leq i \leq m}(x_i, k) \in \Pi$.

A parameterized problem is or-compositional if it has an or-composition algorithm. If the parameterized version of an NP-complete problem admits both a composition and a polynomial kernel, then it also has a distillation. This will imply that if a parameterized problem has a composition algorithm, then it has no polynomial kernel unless $NP \subseteq$ coNP/poly, due to Theorem 2.4.1.

Theorem 2.4.2. ([11]) Let $\Pi(k)$ be an or-compositional parameterized problem such that Π is NP-complete. If Π has a polynomial kernel, then Π also has an or-distillation algorithm.

Algorithms that compose multiple instances of a problem into a single instance have been developed for various problems [11, 70, 12, 10]. Examples of the or-composition technique can be found in [70]: composition by disjoint union, composition using IDs, composition with colors and IDs, and composition as dynamic programming.

Chapter 3 Complexity of And/Or Graph Solution

"They did not know it was impossible, so they did it."

Jean Cocteau

An and/or graph is an acyclic, edge-weighted directed graph containing a single source vertex such that every vertex v has a label $f(v) \in \{\text{and,or}\}$. A solution subgraph H of an and/or-graph must contain the source and obey the following rule: if an and-vertex (resp. or-vertex) is included in H then all (resp. one) of its out-edges must also be included in H. X-y graphs are defined as a natural generalization of and/or graphs. In this chapter we first present the results published in the paper [U. S. Souza, F. Protti, M. Dantas da Silva, Revisiting the complexity of and/or graph solution, J. Comput. Syst. Sci. 79:7 (2013) 1156-1163] where we have investigated the complexity of such problems under various aspects, including parameterized versions of it. However, this article kept the main open question still open: Is the problem of finding a solution subgraph of cost at most k (where k is a fixed parameter) in FPT? We finish this work finally presenting a positive answer to this question, via kernelization techniques. Also, using a framework developed by Bodlaender et al. (2009) and Fortnow and Santhanam (2011), based upon the notion of compositionality, we show that the above parameterized problem does not admit a polynomial kernel unless $NP \subseteq coNP/poly$.

In this chapter we consider the complexity of problems involving two important data structures, and/or graphs and x-y graphs. An and/or graph is an acyclic digraph containing a source (a vertex that reaches all other vertices by directed paths), such that every vertex $v \in V(G)$ has a label $f(v) \in \{\text{and,or}\}$. In such digraphs, edges represent dependency relations between vertices: a vertex labeled **and** depends on all of its outneighbors (conjunctive dependency), while a vertex labeled **or** depends on only one of its out-neighbors (disjunctive dependency).

We define x-y graphs as a generalization of and/or graphs: every vertex v_i of an x-y graph has a label x_i - y_i to mean that v_i depends on x_i of its y_i out-neighbors. Given an and/or graph G, an equivalent x-y graph G' is easily constructed as follows: sinks of G are vertices with $x_i = y_i = 0$; and-vertices satisfy $x_i = y_i$; and or-vertices satisfy $x_i = 1$.

In representations of and/or graphs, and-vertices have an arc around its out-edges. Figure 3.1 shows in (a) an example of and/or graph, and in (b) an example of x-y graph.

And/or graphs were used for modeling problems originated in the 60's within the domain of Artificial Intelligence [55, 83]. Since then, they have successfully been applied to other fields, such as Operations Research, Automation, Robotics, Game Theory, and Software Engineering, to model cutting problems [71], interference tests [53], failure dependencies [6], robotic task plans [20], assembly/disassembly sequences [30], game trees [58], software versioning [27], and evaluation of boolean formulas [59]. With respect to x-y graphs, they correspond to the *x-out-of-y model* of resource sharing in distributed systems [5].

In addition to the above applications, special directed hypergraphs named *F*-graphs are equivalent to and/or graphs [40]. An F-graph is a directed hypergraph where hyperarcs are called *F*-arcs (for forward arcs), which are of the form $E_i = (S_i, T_i)$ with $|S_i| = 1$. An F-graph *H* can be easily transformed into an and/or graph as follows: for each vertex $v \in V(H)$ do f(v)=or; for each *F*-arc $E_i = (S_i, T_i)$, where $|T_i| \ge 2$, do: create an and-vertex v_i , add an edge (u, v_i) where $\{u\} = S_i$, and add an edge (v_i, w_j) for all $w_j \in T_i$.



Figure 3.1: (a) A weighted and/or graph; (b) A weighted x-y graph.

In this work, we denote by O_v and I_v , respectively, the subsets of out-neighbors and in-neighbors of a vertex v. Also, $\tau(e)$ denotes the weight of an edge e, and we define the weight of a graph as the sum of the weights of its edges. We assume |V(G)| = n and |E(G)| = m.

The optimization problems associated with and/or graphs and x-y graphs are formally defined below.

Min-and/or

Instance: An and/or graph G = (V, E) where each edge e has an integer weight $\tau(e) > 0$. Goal: Determine the minimum weight of a subdigraph H = (V', E') of G (solution subgraph) satisfying the following properties:

- $s \in V';$
- if a non-sink node v is in V' and f(v) = and then every out-edge of v belongs to E';
- if a non-sink node v is in V' and f(v) =or then exactly one out-edge of v belongs to E'.

MIN-X-Y

Instance: An x-y graph G = (V, E) where each edge e has an integer weight $\tau(e) > 0$. Goal: Determine the minimum weight of a subdigraph H = (V', E') of G satisfying the following properties:

- $s \in V';$
- for every non-sink node v_i in V', exactly x_i of its y_i out-edges belong to E'.

In 1974, Sahni [81] showed that MIN-AND/OR is NP-hard via a reduction from 3-SAT. Therefore, MIN-X-Y is also NP-hard.

There are three trivial cases for which MIN-AND/OR can be solved in polynomial time:

- 1. All vertices of G are and-vertices. In this case, G is the solution subgraph.
- 2. All vertices of G are or-vertices. In this case, the optimal solution subgraph is a shortest path between s and a sink.
- 3. G is a tree (and/or tree). In this case, the weight of the optimal solution subgraph of G, given by c(s), can be obtained in O(n) time via the recurrence relation below:

$$c(v_i) = \begin{cases} 0, \text{if } v_i \text{ is a sink;} \\ \sum_{v_j \in O_{v_i}} (\tau(v_i, v_j) + c(v_j)), \text{ if } f(v_i) = \text{and;} \\ \min_{v_j \in O_{v_i}} \{\tau(v_i, v_j) + c(v_j)\}, \text{ if } f(v_i) = \text{or.} \end{cases}$$

Other three trivial cases of MIN-AND/OR can be listed: if every or-vertex has outdegree one then or-vertices can be converted into and-vertices, and case 1 above applies; if every and-vertex has out-degree one then and-vertices can be converted into or-vertices, and case 2 applies; finally, if every vertex with in-degree greater than 1 is a sink then the recurrence presented in the case 3 can be used.
As noted by Adelson-Velsky in [1], the problem MIN-AND/OR has interesting connections with real-word applications in scheduling. An example is the work [1], which employs and/or graphs to model real-time scheduling of tasks in computer communication systems. Such a scheduling problem (AND/OR-SCHEDULING) generalizes the classical shortest-path and critical-path problems in graphs [1]. Given a weighted and/or graph, AND/OR-SCHEDULING consists of finding the earliest starting times $t(v_i)$, for all $v_i \in V(G)$, satisfying the following conditions:

• $t(v_i) = 0$, if v_i is a sink;

•
$$t(v_i) \ge \max_{v_j \in O_{v_i}} \{ \tau(v_i, v_j) + t(v_j) \}$$
, if $f(v_i) = \text{and}$

• $t(v_i) \ge \min_{v_j \in O_{v_i}} \{ \tau(v_i, v_j) + t(v_j) \}$, if $f(v_i) =$ or.

MIN-AND/OR can thus be viewed as a variant of AND/OR-SCHEDULING: while the latter aims at determining the minimum *time* necessary to perform a task, the former aims at determining the minimum *cost* to perform it. Since AND/OR-SCHEDULING is solvable in polynomial time [1], its solution can be used as a practical lower bound for MIN-AND/OR. In addition, the recurrence equations for and/or trees lead to a bottom-up dynamic programming algorithm to find in polynomial time a feasible solution (and hence an upper bound) of MIN-AND/OR.

An x-y tree is an x-y graph where no two vertices share a common out-neighbor. As for MIN-AND/OR, MIN-X-Y can be solved in O(n) time when the input x-y graph is an x-y tree T = (V, E). To show this, observe first that the minimum weight of a solution subtree is given by a similar recurrence (shown below), since the optimal solution of an x-y tree rooted at a vertex v_i is obtained by x_i subtrees of v_i :

$$c(v_i) = \begin{cases} 0, \text{ if } v_i \text{ is a sink;} \\ \min_{X \subseteq O_{v_i}, |X| = x_i} \left\{ \sum_{x \in X} \left(\tau(v_i, x) + c(x) \right) \right\} \end{cases}$$

For each non-sink v_i , we need to compute the sum of the x_i smallest values $\tau(v_i, x) + c(x)$ among its children; determining the x_i -th smallest value takes $O(y_i)$ time, and thus selecting the x_i smallest values takes $O(y_i)$ time as well. Then the entire bottom-up procedure takes overall $\sum_{i=1}^{n} O(y_i) = O(n)$ time.

Motivated by the large applicability as well as the hardness of MIN-AND/OR and MIN-X-Y, we study new complexity aspects of such problems, both from a classical and a parameterized point of view. The latter is justified by the fact that many applications are concerned with satisfying a low cost limit. The remainder of this chapter is organized as follows. In Section 3.1, we prove that MIN-AND/OR remains NP-hard even for a very

restricted family of and/or graphs where edges have weight one and or-vertices have outdegree at most two (apart from another property related to some in-degrees), and that deciding whether there is a solution subtree with weight exactly k of a given x-y tree is NP-hard. In Section 3.2, we show that: (i) the parameterized problem MIN-AND/OR(k, r), which asks whether there is a solution subgraph of weight at most k where every or-vertex has at most r out-edges with the same weight, is FPT; (ii) the parameterized problem MIN-AND/OR $^{0}(k)$, whose domain includes and/or graphs allowing zero-weight edges, is W[2]-hard; (iii) the parameterized problem MIN-X-Y(k) is W[1]-hard. Finally, we prove that MIN-AND/OR(k) is also fixed-parameter tractable.

3.1 NP-hardness Results

We now consider a very restricted family of and/or graphs, defined as follows: Let \mathcal{F} be the set of all and/or graphs G satisfying the following properties: every edge in E(G) has weight one; every or-vertex in V(G) has out-degree at most two; and vertices in V(G) with in-degree greater than one are within distance at most one of a sink. We show that even for such and/or graphs the problem MIN-AND/OR remains NP-hard.

Theorem 3.1.1. MIN-AND/OR restricted to \mathcal{F} is NP-hard.

Proof. The proof uses a reduction from VERTEX COVER, shown to be NP-hard by Karp in [56]. Given a graph G = (V, E), we construct an and/or graph G' = (V', E') in \mathcal{F} as follows. Suppose $V = \{v_1, \ldots, v_n\}$ and $E = \{e_1, \ldots, e_m\}$. Create a source $s \in V'$ with f(s) =and. For each edge $e_i \in E$ create an out-neighbor $w_{e_i} \in V'$ of s with $f(w_{e_i}) =$ or. For each vertex $v_j \in V$ create a vertex $w_{v_j} \in V'$ with $f(w_{v_j}) =$ or, and add an edge (w_{e_i}, w_{v_j}) in E' if and only if e_i is incident to v_j . Finally, create an out-neighbor t_{v_j} for each vertex $w_{v_j} \in V'$ and assign $\tau(e) = 1$ for all $e \in E'$. Figure 3.2 illustrates in (a) a graph G and in (b) the and/or graph G' obtained by the construction above.

We now show that there is a vertex cover of size at most k in G if and only if there is a solution subgraph of weight at most 2m + k in G'. Suppose first that G has a vertex cover C of size at most k. A suitable solution subgraph H of G' can be obtained as follows. Vertex s must belong to V(H) by definition. Since s is an **and**-vertex, its m out-edges must belong to E(H). But every out-neighbor w_{e_i} of s is an **or**-vertex; then exactly one of its out-edges in G', say (w_{e_i}, w_{v_j}) , must also belong to E(H). We choose edge (w_{e_i}, w_{v_j}) if and only if $v_j \in C$. At this point, at most |C| vertices w_{v_j} belong to V(H). Now each w_{v_j} has exactly one out-neighbor which is a sink; then for each w_{v_j} we add only one additional out-edge of it. Hence H has weight $2m + |C| \leq 2m + k$.

Conversely, suppose that G' contains a solution subgraph H of weight at most 2m + k. By construction, m out-edges of s belong to E(H), and for each vertex w_{e_i} in V(H)exactly one of its out-edges is in E(H). Since each vertex w_{v_i} in V(H) must have one out-neighbor, V(H) contains at most k vertices w_{v_j} . Let X be the subset of vertices of the form w_{v_j} in V(H), and C a subset of vertices of G such that $v_j \in C$ if and only if $w_{v_j} \in X$. Every vertex w_{e_i} in V(H) has an out-neighbor w_{v_j} in V(H), and by construction of G' a vertex w_{e_i} is an in-neighbor of w_{v_j} if and only if e_i is incident to v_j in G. Since every w_{e_i} in V(H) has an out-neighbor $w_{v_j} \in X$, every edge e_i in G is incident to a vertex $v_j \in C$. Hence C is a vertex cover of G and $|C| = |X| \leq k$.



Figure 3.2: A graph G and the corresponding and/or graph G'.

To conclude this section, we show an interesting result concerning x-y trees. Although MIN-X-Y can be solved in linear time when restricted to x-y trees, deciding whether there is a solution subtree with weight exactly k of a given x-y tree is NP-hard. The following problem is useful:

P-SUBSET SUM

Instance: Finite set Z, integer size s(z) > 0 for each $z \in Z$, positive integers p and q. Question: Is there a subset $Z' \subseteq Z$ such that |Z'| = p and $\sum_{z \in Z'} s(z) = q$?

Lemma 3.1.1. The problem P-SUBSET SUM is NP-hard.

Proof. The proof uses a reduction from the SUBSET SUM problem, shown to be NPhard by Karp in [56]. Consider an instance of SUBSET SUM, formed by: (i) a finite set Asuch that |A| = n and each $a \in A$ has an integer size t(a) > 0; (ii) a positive integer b. Construct an instance of P-SUBSET SUM as follows:

- set p = n and q = b + n;
- set $Z = A \cup A'$, where A' is a set consisting of n new elements;
- set s(z) = t(z) + 1 if $z \in A$;

• set s(z) = 1 if $z \in A'$.

It is easy to see that there is a subset $X \subseteq A$ with $\sum_{x \in X} t(x) = b$ if and only if there is a subset $Z' \subseteq Z$ with |Z'| = p and $\sum_{z \in Z'} s(z) = q$.

Theorem 3.1.2. Let T be an x-y tree. Deciding whether there is a solution subtree T' of T with weight exactly k is NP-hard.

Proof. The proof uses a reduction from the P-SUBSET SUM problem. Given a finite set Z, integer sizes s(z) > 0 for each $z \in Z$, and positive integers p, q, we construct an x-y tree T = (V, E) such that there is a solution subtree T' of T of weight exactly k = q + p if and only if there is a subset Z' of Z such that |Z'| = p and the sum of the sizes of the elements in Z' equals q. The construction is as follows. Create a source vertex $v \in V(T)$ with label p-n. For each element $z \in Z$, create a vertex $u_z \in V(T)$ with label 1-1 and add an edge $e_z = (v, u_z) \in E(T)$ where $\tau(e_z) = 1$. Finally, for each element $z \in Z$, create a vertex w_z with label 0-0 and add an edge $f_z = (u_z, w_z)$ with $\tau(f_z) = s(z)$.

Suppose that there is a subset Z' of Z such that |Z'| = p and the sum of sizes of its elements equals q. Since the source vertex v has label p-n, a solution subtree T' is constructed as follows: $v \in V(T')$, and for each $z \in Z'$ add edges (v, u_z) and (u_z, w_z) to E(T'), where u_z and w_z are vertices associated with z by construction. Observe that each out-edge e of v satisfies $\tau(e) = 1$, and each edge $f_z = (u_z, w_z)$ satisfies $\tau(f_z) = s(z)$. Hence the weight of T' is k = q + p.

Conversely, suppose that there is a solution subtree T' of T with weight p + q. By definition, $v \in V(T')$, and there are p out-edges of v belonging to E(T'), each one with weight equal to 1. Let E' be the subset of edges of the form $f_z = (u_z, w_z)$ in E(T'). Note that |E'| = p and $\sum_{f_z \in E'} \tau(f_z) = q$. Define $Z' = \{z \in Z \mid f_z = (u_z, w_z) \in E'\}$. Clearly, |Z'| = p and $\sum_{z \in Z'} z = q$.

3.2 Parameterized Complexity Results

The problem MIN-AND/OR(k, r)

By Theorem 3.1.1, MIN-AND/OR remains NP-hard even when each or-vertex has at most two out-neighbors. Let MIN-AND/OR(k, r) stand for the parameterized version of MIN-AND/OR where every or-vertex of the input graph has at most r out-edges with the same weight and it is asked whether there is a solution subgraph of weight at most k. Note that the restriction "at most r out-edges with the same weight" imposed on orvertices is in fact a far more general situation than simply restricting the out-degree of vertices to a constant. (We observe that, in real applications modeled by and/or-graphs, dependency relations usually lead to graphs with out-degrees bounded by a constant.) In this subsection, we show that MIN-AND/OR(k, r) is in FPT for parameters k and r. **Theorem 3.2.1.** MIN-AND/OR(k, r) is reducible to a problem kernel in time O(m).

Proof. The proof is based on some correct reduction rules that must be applied once in the order given below:

- 1. for each and-vertex v_i , if $\sum_{v_i \in O_{v_i}} \tau(v_i, v_j) > k$ then remove it;
- 2. for each edge $e \in E(G)$, if $\tau(e) > k$ then remove it;
- 3. for every vertex $v_i \neq s$, if the weight of a shortest path from s to v_i is greater than k then remove it;
- 4. if some vertex becomes unreachable from s then remove it;
- 5. remove every vertex which has become a sink;
- 6. remove each and-vertex such that some of its out-edges have been removed;
- 7. repeat rules 5 and 6 while needed.

Let G' be the graph obtained by applying the above reduction rules. The reduction rules have removed only vertices and edges that could not be part of a solution subgraph of maximum weight k in G and vice-versa. Thus, if S is a solution subgraph of weight at most k in G' then S is also a solution subgraph of weight at most k in G. Note that the running time to apply the above reduction rules is O(m), since G is acyclic.

In G' the longest shortest-path from s to a sink has cost at most k, and each vertex has at most kr out-neighbors. Thus, G' will have a maximum number of vertices if: (i) all its non-sink vertices have out-degree equal to kr, (ii) no vertex shares a same out-neighbor with another vertex, and (iii) the cost of the shortest path from s to any sink is k. Hence the number of vertices at distance i from s is at most $(kr)^i$, that is, the total number of the vertices in G' is at most $O((kr)^{k+1})$.

Since (a) the reduction rules can be applied in O(m) time, (b) the size of G' is a function of the parameters k and r, and (c) a solution subgraph of maximum weight k in G' is also a solution subgraph of maximum weight k in G, we conclude that G' is a kernel for MIN-AND/OR(k, r). Hence MIN-AND/OR(k, r) is reducible to a problem kernel in O(m) time.

Corollary 3.2.1. MIN-AND/OR(k, r) is in FPT.

And/or graphs with zero-weight edges

In this subsection, we consider the family \mathcal{Z} of and/or graphs where zero-weight edges are allowed. This can model practical situations in which some decisions can be taken at

no cost, although in the original definition of MIN-AND/OR [81] all edges have positive weights. Let MIN-AND/OR⁰(k) stand for the parameterized version of MIN-AND/OR applied to and/or graphs in \mathcal{Z} , and DOMINATING SET(c) for the W[2]-hard parameterized problem where it is asked whether an input graph Q has a dominating set of size at most c (see [31]).

Theorem 3.2.2. DOMINATING SET(c) is FPT-reducible to MIN-AND/OR⁰(k).

Proof. Given an instance (Q, c) of DOMINATING SET(c), we construct an instance (G, k) of MIN-AND/OR⁰(k) as follows: (a) create a source vertex s in G where f(s) =and; (b) for each vertex $v_i \in V(Q)$, create three associated vertices u_i, w_i, t_i where $f(u_i) =$ or, $f(w_i) =$ and, $f(t_i) =$ or; (c) for each vertex $u_i \in V(G)$, add an edge (s, u_i) with $\tau(s, u_i) = 0$, and add an edge (u_i, w_j) with $\tau(u_i, w_j) = 0$ if and only if i = j or $(v_i, v_j) \in E(Q)$; (d) create an edge $(w_i, t_i) \in E(G)$ with $\tau(w_i, t_i) = 1$ for all $i \in \{1, \ldots, n\}$; (e) finally, set k = c.

If Q contains a dominating set C such that $|C| \leq c$ then it is possible to construct a solution subgraph H of G with weight at most k as follows: s and all of its out-neighbors belong to V(H); for each vertex $u_i \in V(H)$, include in V(H) an out-neighbor w_j of u_i if and only if $v_j \in C$; and for each vertex $w_j \in V(H)$, add an edge (w_j, t_j) to E(H). Since $|C| \leq c = k$ then at most k edges (w_j, t_j) belong to E(H). Hence H has weight at most k.

Conversely, if G has a solution subgraph H with weight at most k then it is possible to obtain a dominating set C of Q as follows: a vertex v_i of Q belongs to C if and only if w_i belongs to V(H). Since H is a solution subgraph, by definition every non-sink or-vertex has exactly one out-neighbor. Hence H has at most k vertices w_i and $|C| \leq k$.

Figure 3.3 illustrates in (a) an instance of DOMINATING SET and in (b) the corresponding instance of MIN-AND/OR⁰(k) obtained by the construction above.



Figure 3.3: (a) A graph G, (b) the corresponding instance of MIN-AND/OR⁰(k).

Corollary 3.2.2. MIN-AND/OR⁰(k) is W[2]-hard.

3.3 The problem MIN-X-Y(k)

Let MIN-X-Y(k) stand for the parameterized version of MIN-X-Y, where it is asked whether there is a solution subgraph of weight at most k, and CLIQUE(c) for the W[1]hard parameterized problem where it is asked whether the input graph Q has a clique of size c (see [31]).

Theorem 3.2.3. CLIQUE(c) is FPT-reducible to MIN-X-Y(k).

Proof. Given an instance (Q, c) of CLIQUE(c), we construct an instance (G, k) of MIN-X-Y(k) as follows:

- create a source vertex s in G;
- create a set $\{u_1, u_2, ..., u_n\}$ of out-neighbors of s, where n = |V(Q)| (vertex u_i of G is associated with vertex v_i in Q);
- for each vertex u_i , create two out-neighbors z_i and w_i of u_i ;
- for each vertex z_i , create an edge (z_i, w_j) if and only if v_i and v_i are neighbors in Q;
- for each vertex w_i , create an out-neighbor t_i of w_i (t_i is a sink);
- if $v_i \in V(Q)$ has degree less than or equal to c-1 then $\tau(s, u_i) = c^2 + 3c + 1$ else $\tau(s, u_i) = 1$; for all other edges in G their weights are 1;
- s has label c-n;
- every vertex u_i has label 2-2;
- every vertex w_i has label 1-1;
- every vertex t_i has label 0-0;
- for each vertex z_i , if $d(v_i) \ge c 1$ then z_i is labeled $(c 1) \cdot d(v_i)$, otherwise z_i is labeled $d(v_i) \cdot d(v_i)$ (where $d(v_i)$ is the number of neighbors of v_i in Q);
- set $k = c^2 + 3c$.

Figure 3.4 illustrates in (a) a graph Q, and in (b) the corresponding graph G.

Observe that the construction of G can be done in O(m) time, since |V(G)| = 4|V(Q)| + 1. We show that Q contains a clique of size c if and only if G contains a solution subgraph of size less than or equal to k.

If Q contains a set of vertices $\{v_1, v_2, ..., v_c\}$ forming a clique C of size c, then a solution subgraph H of G is constructed as follows. Since s is a vertex with label c-n,

choose $\{u_1, u_2, ..., u_c\}$ to be the out-neighbors of s in H. Now each vertex u_i has label 2-2, and thus vertices $w_1, w_2, ..., w_c$ and $z_1, z_2, ..., z_c$ are also part of the solution subgraph H. This implies that vertices $t_1, t_2, ..., t_c$ belong to V(H) as well. At this point, H already contains 4c edges of weight 1. Since each vertex z_i depends on c-1 out-neighbors, choose an out-neighbor w_j of z_i if and only if $v_j \in C$. Note that out-edges of vertices $z_1, z_2, ..., z_c$ add weight c(c-1) to H. In addition, selected out-neighbors of each vertex z_i were already in H before their choice. Hence the weight of H is $c(c-1) + 4c = c^2 + 3c = k$.

Conversely, suppose that G contains an optimal solution subgraph H of weight at most $k \leq c^2 + 3c$. Note that H is a solution subgraph such that: (i) s has c out-neighbors u_i ; (ii) each out-neighbor u_i of s has two out-neighbors z_i and w_i ; (iii) each one of the cvertices z_i has c - 1 out-neighbors. From these observations, H contains so far at least $c^2 + 2c$ edges, that is, H contains at most c vertices w_i . By construction, if $w_i \in V(H)$ then vertices u_i and z_i also belong to V(H); but since there is no edge between z_i and w_i , H contains exactly c vertices w_i , and $(z_i, w_j) \in E(H)$ for all $w_j \neq w_i$ belonging to V(H). Let C be the subset of vertices $v_i \in V(Q)$ such that $v_i \in C$ if and only if $w_i \in H$. Since u_i, z_i, w_i in G are associated with v_i in Q and out-edges of z_i in G represent the neighborhood of v_i in Q, we conclude that C is a clique of size c in Q. Hence CLIQUE(c) is FTP-reducible to MIN-X-Y(k).

Corollary 3.2.3. MIN-X-Y(k) is W[1]-hard.

Figure 3.4: FPT-reduction of graph Q in (a) to x-y graph G in (b).

Souza, Protti and Dantas da Silva [88], in the paper "Revisiting the Complexity of And/Or Graph Solution" (Journal of Computer and System Sciences 79-7, 2013 1156-1163), published the previous results. Table 3.1 illustrates published parameterized results on MIN-AND/OR and its variations.



_

	FPT	W[1]-hard	W[2]-hard
$\operatorname{Min-and}/\operatorname{Or}(k,r)$	Х	—	_
$\operatorname{Min-and}/\operatorname{Or}(k)$?	?	?
MIN-X-Y(k)	_	Х	_
$\mathrm{Min} ext{-and}/\mathrm{Or}^0(k)$	_	_	Х

Table 3.1: Results on variations of MIN-AND/OR(k)

As we can observe, the question of classifying the parameterized problem MIN-AND/OR(k) was open up to now. The following theorem closes this question, using a reduction to a problem kernel.

Theorem 3.2.4. MIN-AND/OR(k) is fixed-parameter tractable.

Proof. The proof is based on the following reduction rules, that must be applied once in the order given below.

The property key of this kernelization consists of restricting to at most k the number of out-neighbors of each **or**-vertex at a distance at most k of the source s. This property is assured by the application of the Rules 1,2,3,4,5,6 and 7. During the application of this rules, whithout loss of generality, auxiliary vertices and edges are created. Edges with the same coloring represent a single edge in the original intance and auxiliary vertices may represent a set of similar vertices in the original graph.

To clarify the kernelization, pictures illustrating steps of the aplication of some rules is shown after the description.

- 1. For every or-vertex, select an edge pointing to a sink with minimum weight (if there exists) and remove all other out-edges pointing to a sink.
- 2. Assign label and to every sink.
- 3. Color each edge e with a distinct color c_e and define the flag of c_e as the weight of e.



Figure 3.5: Example of application of Rule 4a

- 4. For each and-vertex v having one or-in-neighbor, u, with more than k out-edges do:
 - (a) Let E_{vs} = {e_{s1},..., e_{st}} be the set of out-edges of v pointing to a sink. If t > 1 then:
 (i) replace the edges of E_{vs} by a path P of length t from v to a new vertex, where each internal vertex of P is also a new vertex;
 (ii) for each edge of P assign the same color and weight of a distinct edge of E_{vs}.
 - (b) Let A_v = {v₁,..., v_t} be the set of and-out-neighbors of v. If t > 1 then:
 (i) for each vertex v_j ∈ A_v create a new and-vertex w_j; (ii) create the edges (v, w₁) and (w_j, w_{j+1}) for all j < t; (iii) assign the same color and weight of (v, v_j) to the edge pointing to w_j; (iv) for every edge (v_j, r) where v_j ∈ A_v, create an edge (w_t, r) with the same color and weight as (v_j, r); (v) remove all and-out-edges of v distinct from (v, w₁).



Figure 3.6: Example of application of Rule 4b

(c) if v has only one and-out-neighbor v_j and some or-out-neighbors then: (i) create a new and-vertex w and an edge e = (v, w) with the same color and weight as (v, v_j) ; (ii) for every out-neighbor z of v_j create an edge (w, z) with the same color and weight as (v_j, z) ; (iii) for every edge (v, r) where r is an or-vertex, create an edge (w, r) with the same color and weight as (v, r); (iv) remove the out-edges of v distinct from (v, w).



Figure 3.7: Example of application of Rule 4c

(d) if v has only or-out-neighbors then: (i) select an or-out-neighbor w (non-sink) with minimum out-degree; (ii) for each edge (w, w_j), create a new and-vertex w'_j and edges (v, w'_j), (w'_j, w_j), where (w'_j, w_j) has the same color and weight as (w, w_j), and (v, w'_j) has the same color and weight as (v, z) (w ≠ z), create an edge (w'_j, z) (for all j) with the same color and weight as (v, z); (iv) remove all out-edges of v pointing to or-vertices. (v) assign label or to v;



Figure 3.8: Example of application of Rule 4d

- 5. Assign label or to every and-vertex with only one out-neighbor.
- 6. For each or-vertex v with more than k out-edges do: (i) create k or-vertices w_1, \ldots, w_k ; (ii) for each edge (v, v_j) such that $\tau(v, v_j) = i$ and $i \leq k$, create an edge (v, w_i) with weight i and same color as (v, v_j) ; (iii) for each out-neighbor v_j of v such that $\tau(v, v_j) = i$, $i \leq k$, and $v_j \neq w_i$, do: for each edge (v_j, z) create an edge (w_i, z) with the same color and weight as (v_j, z) ; (iv) remove every out-edge of v not created in this step.



Figure 3.9: Example of application of Rule 6

- 7. While there are or-vertices with more than k out-neighbors at a distance at most k from s, do: repeat Rules 4,5 and 6.
- 8. For each and-vertex v_i , if the sum of weights of its out-edges is greater than k then remove it.
- 9. For each edge $e \in E(G)$, if $\tau(e) > k$ then remove it.
- 10. For every vertex v_i , if the weight of a shortest path from s to v_i is greater than k then remove it.
- 11. If some vertex has become unreachable from s then remove it.

- 12. Remove every vertex which has become a sink.
- 13. Remove each and-vertex such that some of its out-edges have been removed.
- 14. Repeat rules 12 and 13 while needed.

At this point, each vertex has at most k out-neighbors and it is at a distance at most k from s, therefore the graph has $O(k^{k+1})$ vertices. On the other hand, the graph may contain a large number of edges due to the existence of parallel edges created by previous rules. Say that two colors assigned to the same subset of edges (by disregarding parallelism of edges) are in the same group of colors. Since the graph has $O(k^{k+1})$ vertices, the number of distinct groups of colors is bounded by a function of k.

15. For each group of colors, select a color with minimum flag and remove all edges (v, w) colored with another color of this group such that v is an or-vertex.

Now, we have obtained a generalized kernelization algorithm from MIN-AND/OR to a variant that allows parallel and colored edges. The following rule returns the kernel to the original problem.

16. As parallel edges and edges with the same color have been created by rules 4,5 and6, apply successively rules 4, 5, 6 *reversibly* until the graph has no parallel edges and only one edge per color.

After applying the rules, the final graph has size bounded by a function of k. Only vertices and edges redundant or not belonging to a solution subgraph of cost at most k have been removed, and a solution subgraph of cost k in this graph implies a solution subgraph of cost k in the original graph. Thus, the above reduction rules obtain a kernel to the problem, i.e., MIN-AND/OR(k) is fixed-parameter tractable.

3.3 Infeasibility of Polynomial Kernels

At this point, it will be demonstrated by the following theorem that MIN-AND/OR does not have polynomial kernel unless $NP \subseteq coNP/poly$.

Theorem 3.3.1. MIN-AND/OR(k) is Or-compositional.

Proof. Suppose the input sequence:

$$q = (G_1, k), (G_2, k), \dots, (G_m, k).$$

It is possible to obtain an output as follows:

- let G' be the disjoint union of all the graphs of q;
- add a new vertex s' to G';
- for each graph G_i , add in G' a 1-weight edge from s' to s_i , where s_i is the source vertex of G_i ;
- set k' as k+1.
- provide the pair (G', k') as output.

As the source vertex of G' is s', clearly G' has a solution subgraph of cost at most k' if and only if there is a graph G_i in q with a solution subgraph of cost at most k.

Corollary 3.3.1. MIN-AND/OR(k) has no polynomial kernel unless $NP \subseteq coNP/poly$ and consequently $PH \subseteq \Sigma_3^p$.

Proof. Follows from Theorems 3.3.1, 2.4.2 and 2.4.1.

3.4 Open Problem

In this chapter we show that MIN-AND/OR(k) is fixed-parameter tractable, but does not have polynomial kernel unless $NP \subseteq coNP/poly$. At this point we ask.

• For which parameters MIN-AND/OR(k) is fixed-parameter tractable and admits polynomial kernelization?

Chapter 4 Flooding Graphs

"Imagination is more important than knowledge."

Albert Einstein

In this chapter, we present new results on flood-filling games, Flood-it and Free-Flood-it. A complete mapping of the complexity of flood-filling games on trees is made, charting the consequences of single and aggregate parameterizations by number of colors, number of moves, maximum distance of the pivot, maximum orbit, number of leaves, and number of "bad moves". Furthermore, we show that Flood-It on trees and Restricted Shortest Common Supersequence (RSCS) are analogous problems, which proves some FPT and W[1]-hard of cases of Flood-it. In addition, we prove that Flood-It remains NP-hard when played on 3-colored trees, which closes an open question. We also present a general framework for reducibility from Flood-It to Free-Flood-It; some NP-hard cases for Free-Flood-It on trees can be derived using this approach. Analyzing the behavior of these games when played on other classes of boards, such as powers of cycles, powers of paths, and circular grids, we describe polynomial time algorithms to play Flood-it on C_n^2 and $2 \times n$ circular grids, and we show that Free-Flood-it is NP-hard on C_n^2 and $2\times n$ circular grids. Finally, we show that Flood-it is fixed-parameter tractable when the size of a minimum vertex cover is a parameter, and it admits a polynomial kernelization if the number of colors is a second parameter.

Flood-It is a one-player combinatorial game, originally played on a colored board consisting of an $n \times m$ grid, where each tile of the board has an initial color from a fixed color set. In the classic game, two tiles are *neighboring* tiles if they lie in the same row (resp. column) and in consecutive columns (resp. rows). A sequence C of tiles is a *path* when every pair of consecutive tiles in C is formed by neighboring tiles. A *monochromatic path* is a path in which all the tiles have the same color. Two tiles a and b are *m*-connected when there is a monochromatic path between them. In Flood-It, a move consists of assigning a new color c_i to the top left tile p (the *pivot*) and also to all the tiles m-connected to p immediately before the move. The objective of the game is to make the board monochromatic ("flood the board") with the minimum number of moves. Figure 4.1 shows a sequence of moves to flood a 3×3 grid colored with five colors.



Figure 4.1: An optimal sequence of moves to flood a 3×3 grid.

A variation of Flood-It is Free-Flood-It, where the player can freely choose which tile will be the pivot of each move. In addition, these games can easily be generalized to be played on any graph with an initial coloring.

Many complexity issues on Flood-It and Free-Flood-It have recently been investigated. In [2], Arthur, Clifford, Jalsenius, Montanaro, and Sach show that Flood-It and Free-Flood-It are NP-hard on $n \times n$ grids colored with at least three colors. Meeks and Scott [67] prove that Free-Flood-It is solvable in polynomial time on $1 \times n$ grids and on 2-colored graphs, and also that Flood-It and Free-Flood-It remain NP-hard on $3 \times n$ grids colored with at least four colors. Up to the authors' knowledge, the complexity of Flood-It on $3 \times n$ grids colored with three colors remains as an open question. Clifford, Jalsenius, Montanaro, and Sach present in [26] a polynomial-time algorithm for Flood-It on $2 \times n$ grids. In [68], Meeks and Scott show that Free-Flood-It remains NP-hard on $2 \times n$ grids. Fleischer and Woeginger [78] proved that Flood-It is NP-hard on trees.

Flood-filling games in bioinformatics. Since the 90's, an increasing number of papers on biological applications have been dealt with as combinatorial problems. Vertex-colored graph problems have several applications in bioinformatics [34]. The Colored Interval Sandwich Problem has applications in DNA physical mapping [36, 43] and in perfect phylogeny [65]; vertex-recoloring problems appear in protein-protein interaction networks and phylogenetic analysis [25, 72]; the Graph Motif Problem [34] was introduced in the context of metabolic network analysis [60]; the Intervalizing Colored Graphs Problem [13] models DNA physical mapping [36]; and the Triangulating Colored Graph Problem [13] is polynomially equivalent to the Perfect Phylogeny Problem [47].

Flood-Filling games on colored graphs are also related to many problems in bioinformatics. As shown in this paper, Flood-It played on trees is analogous to a restricted case of the Shortest Common Supersequence Problem [48]. Consequently, these games inherit from the Shortest Common Supersequence Problem many applications in bioinformatics, such as: microarray production [79], DNA sequence assembly [7], and a close relationship to multiple sequence alignment [82]. In addition, some disease spreading models, described in [3], work in a similar way to flood-filling games.

Additional definitions and notation.

- Neighboring tiles naturally correspond to neighboring vertices of a graph G representing the board; therefore, from now on, we use the term *vertex* instead of *tile*. A subgraph H of G is *adjacent* to a vertex $v \in V(G)$ if v has a neighbor in V(H).
- A flood move, or just move, is a pair m = (p, c) where p is the pivot of m (the vertex chosen to have its color changed by m), and c is the new color assigned to p; in this case, we also say that color c is played in move m. In Flood-It all moves have the same pivot.
- A subgraph *H* is said to be *flooded* when *H* becomes monochromatic. A vertex *v* is *flooded by a move m* if the color of *v* is played in *m* and *v* becomes m-connected to new vertices after playing *m*. We say that a move *m floods a vertex v by a vertex w* if *v* and *w* are neighbors and move *m* changes the color of *w* to flood *v*.
- A (free-)flooding is a sequence of moves in (Free-)Flood-It which floods G (the entire board). An optimal (free-)flooding is a flooding with minimum number of moves.
- A move m = (p, c) is played on subgraph H if $p \in V(H)$.
- A monochromatic subgraph H' of a subgraph H is abbreviated a mcs of H.
- An *island* is a vertex v colored with a color c such that no neighbor of v is colored with c.
- Let G_n be a graph with n vertices, the k-th power of G_n , denoted by G_n^k , is the graph formed by G_n plus edges between vertices at a distance at most k. Thus, P_n^k and C_n^k is the k-th power of a path P_n and a cycle C_n , respectively.
- A *circular grid* is an $n \times m$ grid with the additional property that the first and the last tiles in a same row are neighboring tiles.
- We denote by $\Pi \propto^{f} \Pi'$ a reduction from a problem Π to a problem Π' via a computable function f.

We present below the formal definitions of the two flood-filling games studied in this chapter.

Flood-It (decision version)

Instance: A colored graph G with a pivot vertex p, an integer λ .

Question: Is there a sequence of at most λ flood moves which makes the graph monochromatic, using p as the pivot in all moves?

Free-Flood-It (decision version)

Instance: A colored graph G with a pivot vertex p, an integer λ . Question: Is there a sequence of at most λ flood moves which makes the graph monochromatic?

It is easy to see that both problems belong to NP. Therefore in the proofs of NPcompleteness presented in this paper will be demonstrated only the NP-hardness of the problems.

Parameters of flood-filling games on trees.

Definition 4.0.1. Let Π be a flood-filling game and let $S = \{s_1, \ldots, s_n\}$ be a subset of the aspects of Π . $[S_1]$ - $\Pi(S_2)$ is the family of parameterized problems where the aspects in $S_1 \subseteq S$ are fixed constants and the aspects in $S_2 \subseteq S$ are aggregate parameters.

As an example, [d]-Flood-It(c) is the family of parameterized problems where d is a fixed constant and c (number of colors) is the parameter.

In this chapter, we develop a multivariate investigation of the complexity of Flood-It and Free-Flood-It when played on trees. We analyze the complexity consequences of parameterizing flood-filling problems in various ways. We consider the following aspects of the problem:

- c number of colors
- λ number of moves
- d maximum distance of the pivot
- o maximum orbit
- k number of leaves
- r number of bad moves, $r = (\lambda c)$

Given a vertex-colored tree T, the *orbit* of a color b in T, o_b , is the number of occurrences of b in T. We say the the maximum orbit of a vertex-colored tree T is the maximum orbit of a color used in T. A good move for a color c_a is a move that floods all non-flooded vertices with color c_a . A move that is not good is a *bad move*. As in Free-Flood-It there is no fixed pivot, for such a game the parameter d stands for the diameter of the graph.

Our results. In Section 2, we prove that Flood-It remains NP-hard on trees whose maximum orbit is 4 and whose leaves are at distance at most d = 2 from the pivot. Furthermore, we show that Flood-It is in FPT when parameterized by the number of colors c in such trees. Also in Section 2 we show that Flood-It on trees and Restricted Shortest Common Supersequence (RSCS) are analogous problems, in the sense that they can be translated from one to another, keeping complexity features; this implies that Flood-It on trees inherits several complexity results already proved for RSCS, such as some interesting FPT and W[1]-hard cases. In addition, we prove that Flood-It played

on general graphs can be solved in polynomial time when considering r as the parameter. Restricting attention to trees where each symbol occurs at most once in any path from the pivot to a leaf (each such path is analogous to a phylogenetic sequence [35]), we prove the NP-hardness of Flood-It even when restricted to such trees, and the fixed-parameter tractability of the game played on such trees when considering the number of bad moves or the maximum orbit as a parameter. In addition, we prove that Flood-It remains NP-hard when played on 3-colored trees, which closes an open question. We conclude Section 2 by introducing a new variant of Flood-It, called Multi-Flood-It, where each move is played on a set of fixed pivots; we consider Multi-Flood-It played on trees where the pivots are the leaves, and derive some complexity results.

In Section 3, we present a general framework for reducibility from Flood-It to Free-Flood-It, by defining a special graph operator ψ such that Flood-It played on a graph class \mathscr{F} is reducible to Free-Flood-It played on the image of \mathscr{F} via ψ . An interesting particular case occurs when \mathscr{F} is closed under ψ (for instance, trees are closed under ψ). Some NP-hard cases for Free-Flood-It on trees can be derived using this approach. We conclude Section 3 by showing some results on parameterized complexity for Free-Flood-It played on *pc-trees (phylogenetic colored trees)*. A colored rooted tree is a pc-tree if no color occurs more than once in any path from the root to a leaf. We prove that some results valid for Flood-It on pc-trees can be inherited by Free-Flood-It on pc-trees, using another type of reducibility framework.

An extended abstract containing part of this work has previously appeared in [87].

4.1 Flood-it on Trees

We start this section by remarking that Flood-It played on a tree is equivalent to Flood-It played on a rooted tree whose root is the pivot.

Theorem 4.1.1. [d]-Flood-It on trees remains NP-hard when d = 2.

Proof. The proof uses a reduction from the Vertex Cover Problem. We show that there is a vertex cover of size k in a graph G if and only if there is a flooding with n + k moves in the associated tree T. Given a graph G = (V, E) with |V| = n and |E| = m, construct a tree T as follows:

- create a pivot root s with color c_s ;
- for each edge $e_i = uv$ of G, add to T a subset of vertices $E_i = \{u'_i, v'_i, u''_i, v''_i\}$ such that u'_i, v'_i are children of s, v''_i is a child of u'_i , and u''_i is a child of v'_i ;
- define a distinct color c_u for each $u \in V(G)$, and color all vertices of the form u'_i, u''_i (for all *i*) with the color c_u .



Figure 4.2: (a) A graph G; (b) tree T obtained from G.

Figure 4.2 shows a graph G and its associated tree T.

Suppose that G has a vertex cover V' of size k. By construction, the set of vertices not flooded have n colors. By playing n moves (using all the n colors, one for each move), each subset E_i still contains a vertex not m-connected to s. Thus we can play these n moves in the following order: initially, moves played on colors assigned to vertices of V', and then other moves. Since every edge in G contains at least one of its endpoints in V', after these n moves the vertices in T not m-connected to s have colors associated with vertices of V'. Since |V'| = k, we will need at most k additional moves, and therefore the flooding will have n + k moves, as required.

Now assume that T has a flooding with n + k moves. Initially, T contains only the color c_s and n other colors forming a set C of colors. Hence, each color of C is played at least once. We divide the colors of C into two groups: the first group is formed by the colors played more than once, and the second group by colors played only once. In order to flood the subset E_i , the first and the last moves played on it are moves played on colors of the first group. Hence, without loss of generality, we can assume that the n + k moves are played in the following order: (a) the first move of all colors in the first group, (b) the moves of the colors in the second group, and (c) the remaining moves of the colors in the first group. Thus, after playing the moves corresponding to (a) and (b) (note that these are n moves, one for each color), one vertex in each subset E_i remains unflooded. In addition, vertices not m-connected to s have k colors. Since each color in T represents a distinct vertex in G and in the construction each E_i is associated with a distinct edge in G, these k colors correspond to a subset of vertices in G of size k that are a vertex cover of G.

Theorem 4.1.2. [d]-Flood-It(c) is in FPT and admits a polynomial kernelization.

Proof. Let T be a tree with n vertices and pivot p. We show below how to find a polynomial kernel (i.e., a kernel whose size is bounded by a polynomial in c) for the problem, in O(n) time. Apply the following kernelization algorithm:

1. set T' = T;

- 2. contract all children of p in T' with color c_i into a single vertex of color c_i . Note that this rule can be applied since the contracted vertices will always be flooded by the same move in T;
- 3. recursively repeat the previous step for each non-leaf child of p in T'.

After applying the above algorithm, each vertex in T' has at most c children. Thus, T' has at most c^d vertices and is a polynomial kernel for the problem.

Corollary 4.1.1. [o, d]-Flood-It on trees remains NP-hard even when o = 4 and d = 2.

Proof. Garey, Johnson, and Stockmeyer proved that Vertex Cover remains NP-complete even for cubic graphs [42]. By restricting the reduction presented in Theorem 4.1.1 to cubic graphs, we can also conclude that Food-it remains NP-hard even on trees whose maximum orbit is equal to six (o = 6) and whose leaves are at distance at most two from the pivot (d = 2). To decrease the maximum orbit to 4, just contract all children of s with color c_i into a single vertex of color c_i .

When Flood-It is played on trees, there are classes of equivalent trees. Given two instances T_i and T_j of Flood-It on trees, they are *equivalent* if and only if every flooding for T_i is also a flooding for T_j , and vice-versa. In other words, T_i and T_j are equivalent if and only if by applying the algorithm presented in Theorem 4.1.2 to T_i and T_j in order to obtain T'_i and T'_j , and next contracting the maximal monochromatic subgraphs of T'_i and T'_j , the resulting trees are identical.

4.1.1 Analogous Problems

Definition 4.1.1. Two optimization problems Π and Π' are said to be analogous if there exist linear-time reductions f, g such that:

- 1. $\Pi \propto^f \Pi'$ and $\Pi' \propto^g \Pi;$
- 2. every feasible solution s for an instance I of Π implies a feasible solution s' for f(I)such that size(s) = size(s');
- 3. every feasible solution s' for an instance I' of Π' implies a feasible solution s for g(I') such that size(s') = size(s).

Next we have an equivalent definition for decision problems. Denote by $Y(\Pi)$ the set of all instances I of Π yielding a yes-answer for the question " $I \in Y(\Pi)$?".

Definition 4.1.2. Two decision problems Π and Π' in NP are said to be analogous if there exist linear-time reductions f, g such that:

1. $\Pi \propto^f \Pi'$ and $\Pi' \propto^g \Pi;$

- 2. every easy checkable certificate C for the yes-answer of the question " $I \in Y(\Pi)$?" implies an easy checkable certificate C' for the yes-answer of the question " $f(I) \in Y(\Pi')$?" such that size(C) = size(C');
- 3. every easy checkable certificate \mathcal{C}' for the yes-answer of the question " $I' \in Y(\Pi')$?" implies an easy checkable certificate \mathcal{C} for the yes-answer of the question " $g(I') \in Y(\Pi)$?" such that $size(\mathcal{C}') = size(\mathcal{C})$.

Definition 4.1.3. Let Π and Π' be analogous decision problems. The parameterized problems $\Pi(k_1, \ldots, k_t)$ and $\Pi'(k'_1, \ldots, k'_t)$ are said to be p-analogous if there exist FPT reductions f, g and a one-to-one correspondence $k_i \leftrightarrow k'_i$ such that:

- 1. $\Pi(k_1, \ldots, k_t) \propto^f \Pi'(k'_1, \ldots, k'_t)$ and $\Pi'(k'_1, \ldots, k'_t) \propto^g \Pi(k_1, \ldots, k_t);$
- 2. every easy checkable certificate C for the yes-answer of the question " $I \in Y(\Pi(k_1, \ldots, k_t))$?" implies an easy checkable certificate C' for the yes-answer of the question " $f(I) \in Y(\Pi'(k'_1, \ldots, k'_t))$?" such that $k'_i = \varphi'_i(k_i)$ for some function φ'_i $(1 \le i \le t)$;
- 3. every easy checkable certificate \mathcal{C}' for the yes-answer of the question " $I' \in Y(\Pi'(k'_1, \ldots, k'_t))$?" implies an easy checkable certificate \mathcal{C} for the yes-answer of the question " $g(I') \in Y(\Pi(k_1, \ldots, k_t))$?" such that $k_i = \varphi_i(k'_i)$ for some function φ_i $(1 \leq i \leq t)$.

Two easy consequences of the above definitions are: (a) if Π and Π' are analogous problems then Π is in P (is NP-hard) if and only if Π' is in P (is NP-hard); (b) if $\Pi(k_1, \ldots, k_\ell)$ and $\Pi'(k'_1, \ldots, k'_\ell)$ are p-analogous problems then $\Pi(k_1, \ldots, k_\ell)$ is in FTP (admits a polynomial kernel/is W[1]-hard) if and only if $\Pi'(k'_1, \ldots, k'_\ell)$ is in FTP (admits a polynomial kernel/is W[1]-hard).

Fleischer and Woeginger used a reduction from the Fixed Alphabet Shortest Common Supersequence Problem [78] to prove that Flood-It on trees is NP-hard even when the number of colors is fixed.

We show that Flood-It on trees and Restricted Shortest Common Supersequence (RSCS) are analogous problems. RSCS is a variant of SCS - Shortest Common Supersequence [35].

Shortest Common Supersequence (SCS)

(decision version)

Instance: A set of strings $S = s_1, \ldots, s_\ell$ over an alphabet Σ , an integer Λ .

Question: Does there exist a string $s \in \Sigma$ of length at most Λ that is a supersequence of each string in S?

Restricted Shortest Common Supersequence (RSCS)

(decision version)

Instance: A set of ρ -strings $R = r_1, \ldots, r_\ell$ over an alphabet Σ , an integer Λ . (A ρ -string is a string with no identical consecutive symbols.)

Question: Does there exist a string $r \in \Sigma$ of length at most Λ that is a supersequence of each ρ -string in R?

Let $SCS(|\Sigma_1|, \ell_1)$ stand for the SCS problem parameterized by $|\Sigma_1|$ and ℓ_1 (ℓ_1 is the number of strings). The notation $RCSC(|\Sigma_2|, \ell_2)$ is used similarly.

Theorem 4.1.3. $SCS(|\Sigma_1|, \ell_1)$ is FPT-reducible to $RSCS(|\Sigma_2|, \ell_2)$.

Proof. Let I be an instance of $SCS(|\Sigma_1|, \ell_1)$. Create an instance I' of problem $RSCS(|\Sigma_2|, \ell_2)$ as follows: for each string s_i of I define a ρ -string r_i of I' by inserting a new symbol c_i after each symbol of s_i . After this construction, I' contains $\ell_2 = \ell_1$ ρ -strings over an alphabet Σ_2 such that $|\Sigma_2| = |\Sigma_1| + \ell_1$. At this point, it is easy to see that I contains a supersequence of length L if only if I' contains a supersequence of length $|s_1| + \ldots + |s_{\ell_1}| + L$.

Theorem 4.1.4.

- (a) Flood-It on trees and RSCS are analogous problems.
- (b) Flood-It(c, k, λ) on trees is p-analogous to $RSCS(|\Sigma|, \ell, \Lambda)$.

Proof. We prove only item (a), the proof of (b) is similar. Given an instance I of RSCS, we create a colored tree T as follows: (i) each position of a string in I is converted into a vertex of T; (ii) if a position of a string in I contains a character c_i then the corresponding vertex of T receives color c_i ; (iii) an edge is added between two vertices of T if and only if they represent consecutive positions of the same string; (iv) a pivot vertex p is created in T with a new color; (v) an edge (p, v) is added to T if v represents the first position of a string in I.

After this construction, note that if I admits a supersequence of length L then T has a flooding with L moves, obtained by traversing the supersequence and playing color c_i in the j^{th} -move if character c_i is in the j^{th} -position of the supersequence. Similarly, given a flooding F of T with k moves, we can construct a supersequence s of length k for I, by just adding character c_i in position j of s if and only if color c_i is played in the j^{th} -move of F.

On the other hand, given an instance T of Flood-It on trees, we create an instance I of RSCS as follows: (i) each color in T is associated with a character of the alphabet Σ over which strings in I are defined; (ii) for each path P from the pivot to a leaf in T, a string r(P) of I is created by first contracting each maximal monochromatic subgraph of P into a single vertex with the same color, and then by adding character c_i in the

 j^{th} -position of r(P) if the vertex in P at distance j > 0 from the pivot has color c_i . If T has a flooding with k moves then I admits a supersequence of length k, since, in order to flood a leaf, one needs to flood the path that connects it to the pivot. As previously, if I admits a supersequence of length L then T has a flooding with L moves, obtained by traversing the supersequence and playing color c_i in the j^{th} -move if character c_i is in the j^{th} -position of the supersequence.

Observation. By Theorem 4.1.2 and Theorem 4.1.4(b), the problem RSCS parameterized by $|\Sigma|$, where ρ -strings have length bounded by a constant l, admits a polynomial kernel, namely, a data structure known as *trie* [39] constructed from the input ρ -strings. Such a data structure is a prefix tree with at most $|\Sigma|^l$ nodes.

Corollary 4.1.2. Flood-It on paths with arbitrary pivot is analogous to RSCS for $k \leq 2$.

By Theorem 4.1.4, results valid for RSCS can be inherited by Flood-It on trees:

Corollary 4.1.3. [k]-Flood-It on trees is solvable in polynomial time.

Proof. Follows from Theorem 4.1.4(a) and the analogous result in [64]: SCS (and thus RSCS) is solvable in polynomial time for a constant number of strings.

Corollary 4.1.4. Flood-It(k, c) on trees is W[1]-hard.

Proof. Follows from Theorems 4.1.3, Theorem 4.1.4(b), and the analogous result in [77]: $SCS(|\Sigma_1|, k_1)$ is W[1]-hard.

4.1.2 Phylogenetic Colored Trees

Flood-It played on trees can be applied to scheduling. Each color corresponds to an operation in the sequential process of manufacturing an object. In the input tree T, paths from the pivot to the leaves correspond to the manufacturing sequences for a number of different objects that share the same production line. A flooding to T then corresponds to a schedule of operations for the production line that allows all of the different objects to be manufactured. It may reasonably be the case that each object to be manufactured requires any given operation to be applied at most once.

Restricting attention to phylogenetic colored trees, we have significant effects on problem complexity.

Theorem 4.1.5. [r]-Flood-It on general graphs can be solved in polynomial time.

Proof. There are c^r possibilities of arrangement of colors for the sequence of r bad moves. Given a sequence of r bad moves, in order to check whether it is possible to flood the input graph using these bad moves, just before and after each bad move apply all possible good moves.

Definition 4.1.4. A colored rooted tree is a pc-tree (phylogenetic colored tree) if no color occurs more than once in any path from the root to a leaf.

Corollary 4.1.5. Flood-It on trees remains NP-hard even when restricted to pc-trees with pivot root.

Proof. Follows from Theorem 4.1.4(a) and the analogous result in [35]: SCS (and also RSCS) is NP-hard even for strings where no symbol occurs more than once.

Corollary 4.1.6. Flood-It(k) on pc-trees with pivot root is W[1]-hard.

Proof. Follows from Theorem 4.1.4(b) and the analogous result in [35]: SCS (and also RSCS) restricted to strings where no symbol occurs more than once is W[1]-hard when parameterized by the number of strings. ■

Definition 4.1.5. A pc-tree T is a cpc-tree (complete pc-tree) if each color occurs exactly once in any path from the root to a leaf.

Cpc-trees are a special subclass of pc-trees. Many hard cases of Flood-It on pc-trees are easy to solve when restricted to cpc-trees; for example, while Flood-It on pc-trees remains NP-hard when d is the parameter, Flood-It on cpc-trees is trivially solved in FPT time.

As in biological applications the phylogenetic sequences are often complete, the complexity of flood-filling games for complete pc-trees is an interesting issue.

Theorem 4.1.6. Flood-It on trees remains NP-hard even when restricted to cpc-trees with pivot root.

Proof. The proof is based on modifying the construction shown in Theorem 4.1.1. Let T be a tree obtained as in Theorem 4.1.1. We can construct a cpc-tree T' from T as follows:

- initially, set T' = T;
- replace each edge $e_i = (p, v_i)$ (where p is the pivot) by a path g_i containing 3n + k 1 vertices;
- add an edge from p to the first vertex of g_i , and another from v_i to the last neighbor of g_i ;
- Let $\{c_1, c_2, \ldots, c_n\}$ be the set of colors in T (disregarding the pivot color). For each path $g_i = w_1, w_2, \ldots, w_{3n+k+1}$, assign to $w_1, w_2, \ldots, w_{n-2}$ the colors c_1, c_2, \ldots, c_n , respectively (disregarding the color of v_i and its children);
- For j = n 1, ..., 3n + k 1, assign to the vertices in level j + 1 a new color c_j .

Figure 4.3 illustrates the cpc-tree T' obtained from the pc-tree presented in Figure 4.2(b).

Let G be the graph from which T was constructed. It is easy to see that G has a vertex cover of size k if and only if T' has a flooding of size 4n + 2k + 1.



Figure 4.3: Cpc-tree T' obtained from the pc-tree presented in Figure 4.2(b).

In Theorem 4.1.1 we proved that Flood-It remains NP-hard even when restricted to pc-trees with maximum orbit 4. Theorem 4.1.7 shows that Flood-It on cpc-trees can be solved in polynomial time when considering the maximum orbit as the parameter.

Lemma 4.1.1. Sibling leaves of a cpc-tree have the same color.

Proof. Let w be an internal vertex of a cpc-tree T with two leaf children v and u, and assume that v and u have distinct colors. As T is a cpc-tree, the color of u occurs once in the path from the pivot to v. Consequently, in the path from the pivot to u, the color of u occurs twice, which is a contradiction.

Theorem 4.1.7. [o]-Flood-It on cpc-trees can be solved in polynomial time.

Proof. Given a cpc-tree T, we first apply to T the algorithm presented in Theorem 4.1.2, obtaining a reduced tree T'. If T' has at most two colors, clearly the maximum orbit is equal to the number of leaves (o = k). Now, assume that for every reduced cpc-tree with j - 1 colors the maximum orbit is equal to the number of leaves. Now we prove that if T' has j colors then the maximum orbit is equal to the number of leaves, in the following way: select a color c_i with minimum orbit (different than the pivot color); for each vertex

v with color c_i , remove it and add edges between the parent of v and the children of v. After this step, T' is a cpc-tree with j-1 colors, and consequently o = k. By Lemma 4.1.1, we observe that returning the removed vertices does not increase the number of leaves, and T' has o = k leaves. Hence, by Corollary 4.1.3, the theorem follows.

4.1.3 Weighted-Flood-it

At this point, we define a natural generalization of Flood-It where each color c_i of the board has cost $\omega(c_i)$ to be played, and we search for the flooding which minimizes the sum of the costs of its moves.

Weighted-Flood-It (decision version)

Instance: A graph G colored over a set C of colors, a pivot vertex $p \in V(G)$, an integer K, and a cost function $\omega : C \to Z^+$.

Question: Does there exist a sequence of moves S with cost at most K which makes the graph monochromatic, using p as the pivot in all moves? The cost of a sequence Sis defined as:

$$\omega(S) = \sum_{m_{c_i} \in S} (\omega(c_i))$$

where m_{c_i} is a move played on color $c_i \in C$.

Corollary 4.1.7. Weighted-Flood-It on cpc-trees with pivot root is in FPT when it is asked whether there is a flooding with cost at most c + K, where c is the number of colors and K is the parameter.

Proof. Follows from Theorem 4.1.4(b) and the analogous result in [35]: SCS (and thus RSCS) is in FPT when every symbol occurs exactly once in each string, and the question is whether there is a common supersequence of cost bounded by $|\Sigma| + K$, where K is the parameter.

Corollary 4.1.8. Flood-It(r) on cpc-trees with pivot root is in FPT.

Proof. Follows from Corollary 4.1.7.

4.1.4 Flood-it on 3-colored Trees

Flood-It on 2-colored graphs is trivially solvable. Fleischer and Woeginger [78] proved that Flood-It remains NP-hard when restricted to 4-colored trees. Raiha and Ukkonen [80] proved that Shortest Common Supersequence over a binary alphabet is NP-complete, and Middendorf [69] proved that Shortest Common Supersequence over a binary alphabet remains NP-complete even if the given strings have the same length and each string contains exactly two ones. In Middendorf's proof the instances of Shortest Common Supersequence do not have two consecutive ones; hence, without loss of generality, we can assume that the last character of each input string is '0' (since after each '1' there is a '0'). We can then define the following NP-complete problem:

q-Shortest Common Supersequence over $\{0,1\}$ (q-SCS)

(decision version)

Instance: A set of strings $S = s_1, \ldots, s_\ell$ over the alphabet $\{0, 1\}$, all ending in '0', and containing exactly two non-consecutive ones, and two integers Λ and q ($q \leq \Lambda - 2$). Question: Does there exist a string s of length at most Λ that is a supersequence of each string in S and contains at least q zeros?

Now we have all the elements to state the following theorem:

Theorem 4.1.8. [c]-Flood-It on trees remains NP-hard when c = 3.

Proof. We use a reduction from q-SCS. Given an instance I of q-SCS, we construct an instance I' of RSCS by adding a new string r of size q composed only by the character '0', and then inserting for every string the character '2' after each occurrence of a '0'.

If for I there is a supersequence of length Λ containing q zeros then there is a supersequence of length $\Lambda + q$ for I'. If for I' there is a supersequence s' of length $\Lambda + q$ then s' contains, say, $\Lambda + q - 2q'$ ones and q' zeros. Removing from s' the occurrences of '2' we obtain a supersequence s for I containing $\Lambda + q - 2q'$ ones and q' zeros. As r has q zeros then $q' \geq q$, and consequently s is a supersequence for I of length less than or equal to Λ and containing at least q zeros.

Given the instance I' of RSCS, an associated 3-colored tree can be constructed assigning the color '2' to the pivot.

Theorem 4.1.9. [c]-Flood-It(d) on trees is in FPT when c = 3.

Proof. Given a 3-colored tree whose leaves are at distance at most d from the pivot, just apply the algorithm presented in Theorem 4.1.2 to obtain T', and next contract the maximal monochromatic subgraphs of T'. The resulting 3-colored tree is a kernel of size 2^d .

4.1.5 Multi-Flood-it on Trees

In this subsection we deal with a new variant of Flood-It, *Multi-Flood-It*, where each move is played on a set of fixed pivots. We assume that, before a move m, all the pivots have the same color. The effect of playing a color c_i in move m is assigning c_i to the pivots and to every vertex m-connected to some pivot immediately before playing m.

We consider Multi-Flood-It played on trees where the pivots are precisely the leaves, called *Multi-Flood-It on trees* for short.

Theorem 4.1.10. Flood-It on trees with pivot root is reducible to Multi-Flood-It on trees.

Proof. Let T be an instance of Flood-It on trees with pivot root. We create an instance T' of Multi-Flood-It on trees with leaf pivots as follows:

- for each path p_i in T from the root to a leaf l_i do:
 - 1. create two copies p_i^1 and p_i^2 of p_i , keeping the same colors of the vertices in p_i ,
 - 2. let r_i^j and l_i^j denote, respectively, the copy of the root of T in p_i^j and the copy of l_i in p_i^j , j = 1, 2,
 - 3. add edge $(l_i^1, l_i^2);$
- contract the vertices r_i^1 (for all *i*) into a single vertex *r*;
- create a new vertex u with the same color as r, and add edge (u, r).

At this point, it is easy to see that T has a flooding of size λ using the pivot root if and only if T' has a flooding of size λ using the leaf pivots.

Corollary 4.1.9. Multi-Flood-It on trees is NP-hard.

Proof. Follows from Theorems 4.1.1 and 4.1.10.

Corollary 4.1.10. Multi-Flood-It(k, c) on trees is W[1]-hard.

Proof. From Theorem 4.1.10 it is easy to see that Flood-It on trees and Multi-Flood-It on trees, both parameterized by the number of leaves and number of colors, are p-analogous. Thus, by Corollary 4.1.4, the result follows.

Theorem 4.1.11. Multi-Flood-It(k, r) on trees is in FPT.

Proof. Suppose the color c_a is chosen for a move of the game. For each path p_i from a leaf l_i to the root, one of the following statements must be true: (1) color c_a is the first color of p_i (different from the pivot color) and does not otherwise occur in p_i ; (2) color c_a does not occur in p_i ; (3) color c_a occurs in p_i , but is not the first color. If for a move of the game only (1) and (2) occur, we call this a *good move*. A move that is not good is *bad*. Our algorithm is based on the following straightforward claims:

Claim 1. If at least r bad moves are played then T has a flooding with at least c + r moves.

Claim 2. For any yes-instance of the problem, there is a flooding with at most r bad moves.

As in [35], we can describe an FPT-algorithm based on the method of *search trees* [32]. By Claim 2, if the answer is "yes" then there is a game that completes with no more than r bad moves. The algorithm is as follows:

- (0) The root node of the search tree is labeled with the given input.
- (1) A node of the search tree is expanded by making a sequence of good moves (arbitrarily) until no good move is possible. For each possible nontrivial bad move (i.e., one that floods at least one vertex), create a child node labeled with the set of sequences that result after this bad move.
- (2) If a node is labeled by the set of empty sequences, then answer "yes".
- (3) If a node has depth r in the search tree, then do not expand it any further.

The correctness of the algorithm follows from Claims 1 and 2, and the fact that the sequence of good moves in step (1) can be made in any order without increasing the number of moves. The running time of the algorithm is bounded by $O(k^r n)$.

Corollary 4.1.11. Flood-It(k, r) on trees is in FPT.

Proof. It is easy to see that we can extend the FPT-algorithm in Theorem 4.1.11 to Flood-It on trees with pivot root.

4.1.6 Free-Flood-it on Trees

Theorem 4.1.12. [r]-Free-Flood-It on general graphs can be solved in polynomial time.

Proof. The proof is similar to the proof presented in Theorem 4.1.5.

Now we present a general framework for reducibility from Flood-It to Free-Flood-It.

Definition 4.1.6. Let G be a graph, $v \in V(G)$, and ℓ a positive integer. The graph $\psi(G, v, \ell)$ is constructed as follows: (i) create ℓ disjoint copies G_1, \ldots, G_ℓ of G; (ii) contract the copies v_1, v_2, \ldots, v_ℓ of v into a single vertex v^* .

Definition 4.1.7. Let \mathscr{F} be a class of graphs. Then:

 $\psi(\mathscr{F}) = \{ G \mid G = \psi(G', v, \ell) \text{ for some triple } (G' \in \mathscr{F}, v \in V(G'), \ell > 0) \}.$

Definition 4.1.8. A class \mathscr{F} of graphs is closed under operator ψ if $\psi(\mathscr{F}) \subseteq \mathscr{F}$.

Examples of classes closed under ψ are chordal graphs and bipartite graphs.

Theorem 4.1.13. Flood-It played on \mathscr{F} is reducible in polynomial time to Free-Flood-It played on $\psi(\mathscr{F})$.

Proof. Let G be an instance of Flood-It on \mathscr{F} (with pivot p). Assume |V(G)| = n. We create an instance for Free-Flood-It on $\psi(\mathscr{F})$ by constructing the graph $G' = \psi(G, p, n)$ and coloring a vertex w_i in copy G_i with the same initial color of its corresponding vertex

 $w \in V(G)$. Now we show that there is a flooding for G with at most λ moves if and only if there is a free-flooding for G' with at most λ moves, as follows. First, note that every flooding F for G implies a free-flooding F' for G' with the same number of moves as F, by simply using p^* as the pivot of all moves in F' and repeating the same sequence of colors played in F. Conversely, if there is a flooding F' for G' with at most λ moves, then: (i) If on every subgraph G_i $(1 \le i \le n)$ of G' a move is played that does not change the color of p^* then $\lambda \ge |F'| \ge n$; in this case, it is easy to see that there is a flooding for Gwith at most λ moves, since |V(G)| = n and thus n - 1 moves suffice to flood G. (ii) If there is a subgraph G_i such that every move played on G_i changes the color of p^* then, without loss of generality, the same sequence of colors played in such moves can be used to flood G, using p as a fixed pivot.

Corollary 4.1.12. Let \mathscr{F} be a class of graphs closed under ψ . Then Flood-It played on \mathscr{F} is reducible in polynomial time to Free-Flood-It played on \mathscr{F} .

NP-hardness results valid for Flood-It can be inherited by Free-Flood-It:

Corollary 4.1.13. [d]-Free-Flood-It on trees remains NP-hard even when d = 4.

Proof. Follows from Theorem 4.1.13 and Theorem 4.1.1.

Corollary 4.1.14. Free-Flood-It on cpc-trees is NP-hard.

Proof. Follows from Corollary 4.1.12 and Corollary 4.1.6.

Corollary 4.1.15. [c]-Free-Flood-It on trees remains NP-hard even when c = 3.

Proof. Follows from Corollary 4.1.12 and Theorem 4.1.8.

Theorem 4.1.14. In Free-Flood-It on pc-trees, there always exists an optimal free-flooding which is a flooding with pivot root.

Proof. Let T be a pc-tree with root p. Let h(T) denote the height of T, and let $C = \{c_1, c_2, \ldots, c_k\}$ be the set of colors assigned to the leaves at level h = h(T) (the root is at level 0). We use induction on h(T). The result is clearly valid when h(T) = 1, since the sequence of moves $(p, c_1), (p, c_2), \ldots, (p, c_k)$ is an optimal free-flooding of T which is a flooding with pivot p. Now assume that the result is valid for all pc-trees with height at most h - 1. By induction, there is an optimal free-flooding F' of the subtree T' obtained from T by removing all the leaves at level h in T, such that F' is a flooding with pivot p. Consider the flooding F of T by appending to F' each move (p, c_i) , where c_i is the color of a non-flooded leaf. Then F is an optimal free-flooding of T which is a flooding with pivot p.

The above theorem implies that Flood-It on pc-trees and Free-Flood-It on pc-trees are analogous, and parameterized versions of these problems are p-analogous. Thus:

Corollary 4.1.16. Free-Flood-It(k) on pc-trees is W[1]-hard.

Proof. Follows from Theorem 4.1.14 and Corollary 4.1.6.

Corollary 4.1.17. Free-Flood-It(k, r) on pc-trees with pivot root is in FPT.

Proof. Follows from Theorem 4.1.14 and Corollary 4.1.11.

Corollary 4.1.18. Free-Flood-It(r) on cpc-trees with pivot root is in FPT.

Proof. Follows from Theorem 4.1.14 and Corollary 4.1.8.

4.2 Flood-Filling Games on Power Graphs

Flood-filling games played on graphs is a powerful model for several real world applications, mainly in Bioinformatics. In [87], for instance, Souza, Protti and Dantas da Silva show that Flood-it played on trees is analogous to an important subcase of the *Shortest Common Supersequence* problem, which is a classical problem from the realm of string analysis. Consequently, these games inherit many implications in bioinformatics, such as: microarray production [79], DNA sequence assembly [7], and a close relationship to multiple sequence alignment [82]. In addition, some disease spreading models described in [3] work in a similar way to flood-filling games on general graphs.

Analysing the complexity of Flood-it on non-grid graphs, Fleischer and Woeginger [78] proved that Flood-it (denoted by Honey-Bee-Solitaire) remains NP-hard even restricted to trees or split graphs, but it is polynomial-time solvable on co-comparability graphs. When Flood-it is played on paths $(1 \times n \text{ grids})$ the problem is trivially solvable if the pivot has degree one, however, allowing to the pivot be any vertex of the path, the problem is analogous to Shortest Common Supersequence problem (SCS) for two sequences [87], which is a very well-studied problem that does not have a known linear-time algorithm. Consequently, it is easy to see that Flood-it on cycles can be solved in polynomial time. In [67], Meeks and Scott show that Free-Flood-it on paths can be solved in $O(n^6)$ time. Thus, by the approach of removing the last vertex that will be flooded, we obtain an algorithm to solve Free-Flood-it on cycles in $O(n^7)$ time.

The main goal of this section is to analise the complexity of flood-filling games played on simple structures. We study the complexity of Flood-it and Free-Flood-it on others classes of boards, such as power of paths, powers of cycles and circular grids. We describe polynomial time algorithms to play Flood-it on C_n^2 (the second power of a cycle on *n* vertices), $2 \times n$ circular grids ($2 \times n$ grids where the first and last tiles in a same row are neighboring tiles), $2 \times n$ *d*-boards ($2 \times n$ grids where the *d*th column is monochromatic), and $2 \times n$ circular *d*-boards. We also show that Free-Flood-it is NP-hard on C_n^2 and $2 \times n$ circular grids.

4.2.1 Flood-it on Circular Boards

A *d*-board is an $n \times m$ grid where the *d*th column is monochromatic. Thus Flood-it on a *d*-board consists of playing the game using column *d* as the pivot. Observe that Flood-it on $2 \times m$ *d*-boards is a generalization of Flood-it on $2 \times m$ boards (take d = 1), and Flood-it on paths (where any vertex can be the fixed pivot) is equivalent to Flood-it on $1 \times n$ *d*-boards. We denote by B_l (resp. B_r) of a *d*-board *B* the board composed by *d* and all tiles to the left (resp. right) of *d*.

In this paper, we show a framework based on *d*-boards to solve Flood-it on circular grids.

Lemma 4.2.1. Given a $2 \times n$ d-board B, and vertices $v_l \in B_l$ and $v_r \in B_r$, the minimum number of moves to connect v_l and v_r can be found in $O(n^2)$ time.

Proof. Let L, R be maximum mcs of B_l and B_r containing d, respectively. We can think on L and R as "dynamic subgraphs" in the sense that they modify after each move. Observe that v_l will be flooded by a vertex which is either in the same column of v_l or in a column to the right of a. Therefore, columns to the left of v_l do not need to be analyzed. An analogous reasoning applies to v_r . Thus in order to choose which color must be played in each move, we only need to know the leftmost (resp. rightmost) vertex (or two vertices) of L (resp. R). This set of one or two vertices defines a *configuration* of L (or R).

At this point, we construct a directed acyclic hypergraph H as follows:

- create a vertex for each possible configuration of L or R;
- given configurations S_1, T_1 and S_2, T_2 of L and R, respectively, add a directed edge $(\{S_1, S_2\}, \{T_1, T_2\})$ labeled with color c if by playing color c it is possible to simultaneously change the configuration of L from S_1 to T_1 and the configuration of R from S_2 to T_2 ;
- given configurations S_1, T_1 of L (or R), add a directed hyperedge $(\{S_1\}, \{T_1\})$ labeled with color c if by playing color c it is possible to change the configuration of L (or R) from S_1 to T_1 ;
- collapse the vertices representing the initial configurations of L and R into a single vertex s.

Each hyperedge of H represents a possible move of the game. A hyperedge of the form $(\{S_1, S_2\}, \{T_1, T_2\})$ represents a move connecting vertices of L and R. For example, the hyperedge $(\{\{e\}, \{i\}\}, \{\{c\}, \{j\}\})$ in Figure 4.4 shows that by playing the color green we connect the tiles c and j via e and i. Finding the minimum number of moves to connect v_l and v_r amounts to finding the minimum number of hyperedges needed to construct

paths between s and vertices representing configurations containing v_l and v_r . Since the number of hyperedges of H is $O(n^2)$, where n = V(B), it is easy to see that we can find the minimum number of moves to connect v_l and v_r in $O(n^2)$ time.

Figure 4.4 shows a $2 \times n$ *d*-board *B* and a subgraph of the hypergraph *H* obtained from *B* which contains the minimum moving to connect *a* and *h*.



Figure 4.4: (a) A $2 \times n$ *d*-board *B*. (b) A subgraph of the hypergraph *H* of *B*.

Theorem 4.2.1. Flood-it can be solved in polynomial time on $2 \times n$ d-boards. More precisely, in $O(kn^2)$ time, where k is the number of colors.

Proof. Suppose that B is a $2 \times n$ d-board and k is the number of colors. We say that a tile t on L (resp. R) is marked if it has color c_t and no other tile in the columns strictly to the left (resp. right) of t has the color c_t . A column is marked if it contains a marked tile. As in [26], the key property which holds on $2 \times n$ d-boards is that if the marked tiles are flooded then so is the whole board B. To see this, note that when a marked tile t of color c_t in L (resp. R) is flooded, all other tiles of color c_t in L (resp. R) that have not yet been flooded are to the right (resp. left) of t and therefore adjacent to the flooded region. Hence they will be flooded when t is flooded. Thus, we iteratively ask for the shortest sequence of moves to connect the rightmost non-flooded marked tile of L and R are connected. Hence, the minimum number of moves needed to flood B can be obtained in time $O(kn^2)$.

Theorem 4.2.2. Flood-it can be solved in $O(kn^4)$ time on $2 \times n$ circular d-boards, where k is the number of colors.

Proof. Let $d, c_1, c_2, \ldots, c_{n-1}$ be the columns of a $2 \times n$ circular d-board B. Let S be an optimal sequence of moves to flood B. Let us say that $v \in c_i$ is right-flooded by S if either i = 1 or $i \leq n-2$, and before playing the move $m \in S$ which floods v there is a path from d to v such that every internal vertex of such a path belongs to the maximum mcs containing the pivot and is right-flooded. On the other hand, a vertex v is left-flooded by

S if it is not right-flooded. A column c_i is said to be right-flooded (left-flooded) by S if its two vertices are right-flooded (left-flooded) by S. Note that even by removing all edges connecting right-flooded vertices to left-flooded vertices, the sequence S still floods B.

Let c_i be the right-flooded column with maximum index i, and v_j be a right-flooded vertex lying in a column j such that j is maximum $(j \ge i)$. Let H_r (resp., H_ℓ) be the subgraph induced by the vertices in d plus the right-flooded (resp. left-flooded) vertices. By removing all edges between H_r and H_ℓ , we can naturally define a $2 \times n$ d-board B' (if after removing the edges some column d' is left with only one vertex w, replace d' by a monochromatic column with the same color as w). Note that an optimal flooding of B'corresponds to an optimal flooding of B. Since c_i and v_j are not previously known, we must execute this process for each possibility (there are $O(n^2)$ many of them). Hence, we can solve Flood-it on $2 \times n$ circular d-boards in $O(kn^4)$ time.

Corollary 4.2.1. Flood-it is solvable in polynomial time on $2 \times n$ circular grids.

Proof. Follows from Theorem 4.2.2, by replacing the column c containing the pivot by three consecutive columns c_1, d, c_2 , where c_1 and c_2 are copies of c, and d is a monochromatic column with the same color of the pivot of c.

Lemma 4.2.2. Flood-it on C_n^2 is a particular case of Flood-it on circular grids.

Proof. Let v_1, v_2, \ldots, v_n be the vertices of a graph C_n^2 . Then, by taking circular indices, the neighborhood of v_i is $\{v_{i-2}, v_{i-1}, v_{i+1}, v_{i+2}\}$. We can create a $2 \times n$ circular grid T equivalent to C_n^2 as follows:

- For *n* even: (i) define $q_{a_1}, q_{b_1}, q_{a_3}, q_{b_3}, \ldots, q_{a_{n-1}}, q_{b_{n-1}}$ as the first row; (ii) define $q_{b_n}, q_{a_2}, q_{b_2}, q_{a_4}, q_{b_4}, \ldots, q_{a_{n-2}}, q_{b_{n-2}}, q_{a_n}$ as the second row.
- For *n* odd: (i) define $q_{a_1}, q_{a_3}, q_{b_3}, q_{a_5}, q_{b_5}, \dots, q_{a_{n-2}}, q_{b_{n-2}}, q_{a_n}$ as the first row; (ii) define $q_{b_1}, q_{a_2}, q_{a_4}, q_{b_4}, \dots, q_{a_{n-1}}, q_{b_{n-1}}$ as the second row.

In both constructions tiles q_{a_i} and q_{b_i} (if it exists) receive the same color as v_i . See Figure 4.5.

By assuming that the "component" $\{q_{a_i}, q_{b_i}\}$ (or $\{q_{a_i}\}$, for n odd and i = 2 or i = n) represent vertex v_i , we observe that: (i) for n even, a tile of color c is adjacent to $\{q_{a_i}, q_{b_i}\}$ if and only if v_i has a neighbor of color c; (ii) for n odd, the same property above is valid, except for $\{q_{a_2}\}$ and $\{q_{a_n}\}$; however, both are adjacent to component $\{q_{a_1}, q_{b_1}\}$ that represents the pivot vertex v_1 . Thus, in Flood-it, C_2^n can be represented as a $2 \times n$ circular grid.

Corollary 4.2.2. Flood-it can be solved in polynomial time on C_n^2 .

Proof. Follows from Lemma 4.2.2 and Corollary 4.2.1.


Figure 4.5: (a) $2 \times n$ circular grid for even n; (b) $2 \times n$ circular grid for odd n.

Corollary 4.2.3. Flood-it can be solved in polynomial time on P_n^2 .

Proof. Follows from a similar construction of Lemma 4.2.2 (desconsidering some edges and retorting a few tiles) and Corollary 4.2.2.

4.2.2 Free-Flood-it on Powers of Cycles

Flood-it on paths can be easily solved in $O(n^2)$ time by a dynamic programming [87], and as show in this paper, the problem remains polynomially solvable when played on circular grids, C_n^2 and P_n^2 . Although Free-Flood-it can be solved in polynomial time when played on paths and cycles, in this section, we show that Free-Flood-it is NP-hard when played on C_n^2 , P_n^2 or circular grids.

Theorem 4.2.3. Free-Flood-it remains NP-hard on C_n^2

Proof. This proof uses a reduction from VERTEX COVER. Let Q be a graph formed by vertices x_1, x_2, x_3, x_4 and edges $e_a = (x_1, x_2), e_b = (x_1, x_3), e_c = (x_2, x_4)$. Note that Qcontains a minimum cover formed by x_1 and x_2 ; this cover contains the two endpoints of e_a . It is clear that VERTEX COVER remains NP-hard for all graphs containing Q as an isolated component. Thus let $G_Q = G \cup Q$ be such a graph, where G is a graph with nvertices, m edges and a vertex cover of size k. Clearly, G_Q has a vertex cover of size k+2.

From G_Q we will construct a graph H isomorphic to a 2-power of a cycle.

- for each edge $e_i = (u, v)$ in G_Q create a gadget g_i in H as follows:
 - create vertices $u_1^{e_i}, u_2^{e_i}, u_3^{e_i}, v_1^{e_i}, v_2^{e_i}, v_3^{e_i}$ and edges $(v_3^{e_i}, u_2^{e_i}), (u_2^{e_i}, u_1^{e_i}), (u_1^{e_i}, v_1^{e_i}), (v_1^{e_i}, v_2^{e_i}), (v_2^{e_i}, u_3^{e_i});$
 - create vertices $z_1^{e_i}, z_2^{e_i}, z_3^{e_i}, \ldots, z_m^{e_i}$, edges $(z_j^{e_i}, z_{j+1}^{e_i}), 1 \le j \le m-1$, and edge $(z_1^{e_i}, v_3^{e_i});$
 - create vertices $y_1^{e_i}, y_2^{e_i}, y_3^{e_i}, \dots, y_m^{e_i}$, edges $(y_j^{e_i}, y_{j+1}^{e_i}), 1 \le j \le m-1$, and edge $(y_1^{e_i}, u_3^{e_i});$
 - add an edge between vertices x, y of g_i whenever they are at distance 2;

- color $u_1^{e_i}, u_2^{e_i}, u_3^{e_i}$ with color $c_u; v_1^{e_i}, v_2^{e_i}, v_3^{e_i}$ with color c_v ; and for every $1 \le j \le m$, vertices $z_j^{e_i}$ and $y_j^{e_i}$ with color $c_j^{e_i}$.
- after constructing the gadgets, for all $1 \le i \le m+2$, add $(z_m^{e_i}, y_m^{e_{i+1}})$;
- add edge $(z_m^{e_{m+3}}, y_m^{e_1});$
- add an edge between x and y in H whenever they are at distance 2.

Each gadget g_i of H is divided into three parts: the core, consisting of the vertices $u_1^{e_i}, u_2^{e_i}, u_3^{e_i}, v_1^{e_i}, v_2^{e_i}, v_3^{e_i}$; the z-arm, consisting of $z_1^{e_i}, \ldots, z_m^{e_i}$; and the y-arm, consisting of $y_1^{e_i}, \ldots, y_m^{e_i}$. We denote by g_a, g_b, g_c the gadgets associated with edges e_a, e_b, e_c , respectively. Figure 4.6 shows a gadget according to the construction above. Its core and arms are shown in detail. Figure 4.7 shows in (a) a graph G, in (b) the graph Q, and in (c) the power of cycle obtained from G_Q . Each detailed gadget, g_i , in (c) is equivalent to an edge e_i in $G_Q, i \in \{1, 2, 3, a, b, c\}$.



Figure 4.6: (a) an edge e_i ; (b) gadget corresponding to e_i .

First, we will prove that if G_Q contains a vertex cover V' of size k+2 (i.e., G contains a vertex cover of size k) then the constructed graph H has a free-flooding with $m^2+5m+k+5$ moves.

By construction, in every gadget g_i of H vertices $z_j^{e_i}$ and $y_j^{e_i}$ have the same color. Thus to make the arms of each gadget $g_i \neq g_a$ monochromatic in only m moves, we play one move in its core such that five vertices become colored with the same color. The remaining vertex (with another color) will be associated with a vertex of the cover in G_Q . After this move, we play m moves, starting from the core, to flood the arms. Figure 4.8 illustrates, as described above, a sequence of m + 1 moves to flood the gadget presented in Figure 4.6(b).

At this point, $m^2 + 3m + 2$ moves were played and each gadget $g_i \neq g_a$ has only two colors, the color of its maximum mcs (denote by h_i this mcs) and the color of an island representing a vertex of V'. Figure 4.9(a) illustrates the graph presented in Figure 4.7(c) after these $m^2 + 3m + 2$. Note that none move was played in the gadget g_a .



Figure 4.7: (a) a graph G; (b) the graph Q; (c) the graph C_n^2 obtained from G_Q

By playing m+1 additional moves we create a big mcs H' from the h_i 's, and by playing more m moves we obtain an mcs H'' which contains H' and both arms of g_a . Figure 4.9(b) illustrates the graph presented in Figure 4.7(c) after created H', and Figure 4.9(b) shows the graph after created H''.

Now vertices that do not belong to H'' are vertices of the core of g_a or islands of other gadgets. Since both endpoints of edge e_a lie in a minimum cover of Q, and assuming that the islands in g_b and g_c represent vertices x_1 and x_2 , more two moves suffice to flood g_a , g_b and g_c .

As each gadget in H represent an edge in G_Q , and each color in a core represent a vertex in a edge, by construction, the coloring of the remaining islands represent a vertex cover of G. Considering that these remaining islands represent the minimum vertex cover of G, H can be flooded in at most k final moves. This gives a free-flooding of H in at most $m^2 + 5m + k + 5$ moves.

Conversely, assume that H has an optimal free-flooding S with $m^2 + 5m + k + 5$ moves. For two vertices a, b belonging, respectively, to the z-arm and the y-arm of a gadget g_i , note that a and b can be flooded by the same move m if and only if a and b have the same color c and, immediately before move m, there exists an mcs H' adjacent to a and b with color $c' \neq c$. When such an mcs H' exists we have two cases: (i) H' contains vertices of the core of g_i (H' is of type 1); (ii) H' is not of type 1, but contains vertices of the arms and the core of all the other gadgets (H' is of type 2).



Figure 4.8: A sequence of moves to flood the arms of the gadget presented in Fig. 4(b).

It is easy to see that during the flooding only one gadget of H, say g_f , can contain vertices a, b such that: (1) a, b lie in distinct arms of g_f ; (2) a, b are flooded by the same move, m, played on an mcs of type 2.

Hence, at least m + 2 gadgets forming a collection U contain no pair a, b of vertices as described. In each gadget g_j in U, the minimum number of moves required to create an mcs containing all vertices of its arms is m + 1, where one of the moves exclusively floods vertices in the core of g_j . Since S is an optimal free-flooding, without loss of generality we can assume that these subsequences of m + 1 moves for each gadget in U correspond to the first (m + 2)(m + 1) moves in S. Figure 4.9(a) illustrates the graph presented in Figure 4.7(c) after a possible sequence of $m^2 + 3m + 2$ moves, as described.

After playing these moves, each gadget in U contains an mcs and an island. Now at least m + 1 moves are necessary to join all the m + 2 mcs's created so far into a single one. Thus there still remain m + k + 2 moves to be analyzed. Observe that joining the m + 2 mcs's, we can use only m moves to flood all vertices in the arms of the remaining gadget, g_f , (gadget whith no flooded vertex). Note that g_f must exist to the free-flooding to be optimal.

At this point, let W' be the subset of vertices of H not yet flooded. They either lie in g_f 's core or are islands in the core of other gadgets. As these vertices are flooded in k+2



Figure 4.9: States of the graph in Figure 5(c) during an optimal flooding

moves, since each gadget in H represents an edge of G_Q and vertices lying in the core of a gadget are associated with vertices of G_Q , by construction, the colors of the vertices in W' correspond to a vertex cover of G_Q of size k + 2. This gives a vertex cover of G of size at most k.

Corollary 4.2.4. Free-Flood-it remains NP-hard on P_n^2 .

Proof. First of all, construct a second power of a path, P_n^2 , using a similar construction of Theorem 4.2.3 without the component Q, and desconsidering the first and the last gadgets as neighbors. After that, we similarly can show that P_n^2 has a free-flooding of size $m^2 + 2m + k - 1$ if and only if G has a vertex cover of size k.

Corollary 4.2.5. Free-Flood-it remains NP-hard on $2 \times n$ circular grids.

Proof. Use the construction in Lemma 4.2.2 (for n even) and Theorem 4.2.3.

4.3 The Size of a Minimum Vertex Cover as Parameter

On the parameterized complexity point of view, we analize the complexity of Flood-it game played on graphs with bounded minimum vertex cover. We describe an FPTalgorithm for Flood-it when fixed the size of the minimum vertex cover as parameter. In addition, we present a polynomial kernelization algorithm when besides the minimum vertex cover, it is fixed the number of colors as parameter.

In the literature, there exists some results considering bounded values for different parameters of Flood-it. For instance, Flood-It is NP-hard on $n \times n$ grids colored with at least three colors [2], and it is also NP-hard on trees with diameter at most four [87]. In [87], Souza et al show some parameterized complexity results on Flood-it on trees, such as: Flood-it is W[1]-hard on trees whose the number of leaves and number of colors are parameters. On the other hand, it is easy to verify in $O(k^k)$ time whether Flood-it has a solution of size at most k, and to obtain a kernel of size $O(c^d)$ where c and d are parameters to the number of colors and the diameter of the graph, respectively. At this point, we analyze the parameterized complexity of Flood-it game considering the minimum vertex cover of the board (graph) as parameter.

Theorem 4.3.1. Flood-it on graphs is fixed-parameter tractable, FPT, when parameterized by the size of the minimum vertex cover (k).

Proof. Let G = (V, E) be the board of the game, where V(G) = n and E(G) = m. First of all, we must observe the following statements:

• Enumerating all vertex covers of size at most k can be done in $O(2^k.n)$ time by the method of bounded search trees [31];

- It is possible to flood a vertex cover of a graph G just playing a subset of moves of any flooding of G;
- After to flood a vertex cover of G, each color is played at most one more time in any optimal flooding of G.

In this proof, we assume that a vertex v of G is flooded by a vertex u if and only if u was the first neighbor of v to be flooded in G, if v is neighbor of the pivot p then v is flooded by p. If v is flooded by u we say that u is a *link* to v.

As it is possible to obtain a minimum vertex cover C of G in $O(2^k.n)$ time, the main idea of this proof is to construct in $f(k).n^{O(1)}$ time a subgraph G' of G, consisting of: (i) the pivot vertex p; (ii) a minimum vertex cover C of G; (iii) possible and non-redundant links to flood C in G. The construction of G' is presented below.

- 1. G' = G;
- 2. For each subset of vertices with the same color and the same neighborhood in $V(G') \setminus \{C \cup \{p\}\}$ do: select one vertex and remove all the others.
- 3. Let S be the set of colors of G' that is not in $\{C \cup \{p\}\}\)$. For each subset Q of S consisting of colors with the same number of vertices with the same neighborhood do:
 - select one color $q \in Q$ and remove all vertices colored with a color in $\{Q \setminus \{q\}\}$.

The step 2 of the construction remove of G' some redundant vertices, and the step 3 remove of the graph some redundat colors, because if $q, r \in Q$ and a vertex with color r is a link to flood a vertex v in C then we can construct another flooding with at most the same number of moves using a vertex with color q as link to flood v.

As we can observe, the construction of G' is done in polynomial time. Now, we must analyse the cardinality of G'.

- By the step 2, there are $O(2^k)$ vertices with the same coloring in G'.
- By the step 3, there are $O(2^{k^{2^k}})$ colors in G'.
- Consequently, there are $O(2^{k^{2^{k+1}}})$ vertices in G'.

At this point, we present the following FPT-algorithm to solve Flood-it on graphs parameterized by the size of the minimum vertex cover.

- 1. To obtain a minimum vertex cover C in FPT-time;
- 2. construct the graph G' in polynomial time;

- 3. exhaustively analyse all the possible sequences of moves to flood C in G';
- 4. For each possible flooding F'_i of C in G', create a flooding F_i of G applying F'_i and after, playing each color at most once until to flood G.
- 5. Set $S = min(F_i)$.

As G' has $O(2^{k^{2^{k+1}}})$ vertices, the exhaustive analysis of all the possible sequences of moves to flood C can be done in f(k) time, and consequently S can be obtained in $f(k).n^{O(1)}$ time. By the costruction of G' we can conclude that S is a optimal flooding of G.

4.3.1 Polynomial Kernelization

Since every FPT problem has a kernelization algorithm, it is interesting to study problems which allow kernelization algorithms that reduce instances to a size which is polynomially bounded by the parameter. Such problems are said to have a *polynomial kernelization algorithm*, or a *polynomial kernel*.

Theorem 4.3.2. Flood-it on graphs admits a polynomial kernelization when parameterized by the size of the minimum vertex cover (k) and the number of colors (c).

Proof. First, we must obtain the minimum vertex cover C of the instance G. The graph G is composed by at most four disjont subset of vertices: $\{p\}, C \setminus \{p\}, A$, and B, where A contains the vertices adjacents to p and vertices in $C \setminus \{p\}$ (A can be empty), and B contains the vertices adjacents only to vertices in $C \setminus \{p\}$. Given these definitions we construct a polynomial kernel H of G as follows.

1. set $V(H) = \{p\} \cup C \setminus \{p\} \cup A;$

If a vertex $v \in C$ was flooded by a vertex $x \in B$ then x has a color i and was flooded by a vertex $w \neq v$ and $w \in C$.

- 2. for each vertex $v \in C$, for each vertex $w \in C$ (w can be v), and for each color i do:
 - select, if exists, a vertex $x \in B$ of color i neighbor of v and w and set $V(H) = V(H) \cup \{x\}$
- 3. set H = G[V(H)]
- 4. contract all children of p in H with color c_i into a single vertex of color c_i . Note that this rule can be applied since the contracted vertices will always be flooded by the same move in H;

Note that:

- $|C| \leq k;$
- $A \leq c$ (by rule 4);
- $|B| \le k.(k.c) = ck^2$ (by rule 2);
- $|V(H)| \le ck^2 + c + k + 1.$

If a vertex $z \in B$ not belongs to V(H) then for each pair v, w of neighbors of z there is a vertex $y_i \in V(H)$ neighbor of v, w with the same color than z which also belongs to B. Consequently, any flooding of H is a flooding of G where z will be flooded in the same move than a vertex y_i . As $|V(H)| \leq ck^2 + c + k + 1$, then H is a polynomial kernel.

4.4 Final Considerations

In this work, we first develop a multivariate investigation on the complexity of Flood-It and Free-Flood-It when played on trees, analyzing the complexity consequences of parameterizing flood-filling problems by one or two of the following parameters: c-number of colors; λ - number of moves; d - maximum distance of the pivot (or diameter, in the case of Free-Flood-It); o - maximum orbit; k - number of leaves; r - number of bad moves. During our analysis we have shown that Flood-It on trees is analogous to Restricted Shortest Common Supersequence, and Flood-It remains NP-hard on 3-colored trees, closing an open question. We also present a general framework for reducibility from Flood-It to Free-Flood-It. The obtained results are summarized in the table below. The trivial cases presented in the table easily remain fixed-parameter tractable for Free-Flood-It.

COMPLEXITY OF FLOOD-FILLING GAMES ON TREES					
Problem	Instance	Fixed constant	Parameters	Complexity	Reference
Flood-It	cpc-trees		_	NP-hard	Theor. 4.1.6
	trees	$c \ge 3$	_	NP-hard	Theor. 4.1.8
	pc-trees		с	FPT	Trivial
	graphs		λ	FPT	Trivial
	cpc-trees		d	FPT	Trivial
	cpc-trees	0	_	Polynomial	Theor. 4.1.7
	pc-trees		k	W[1]-hard	Corol. 4.1.6
	graphs	r	_	Polynomial	Theor. 4.1.5
	cpc-trees		r	FPT	Corol. 4.1.8
	trees		c, d	FPT	Theor. 4.1.2
	graphs		c, o	FPT	Trivial
	trees		c,k	W[1]-hard	Corol. 4.1.4
	trees		c, r	FPT	Trivial
	pc-trees	$d\geq 2,o\geq 4$	—	NP-hard	Coroll. 4.1.1
	trees		d,k	FPT	Trivial
	trees		o,k	W[1]-hard	Corol. 4.1.6
	trees		k,r	FPT	Corol. 4.1.11
Free-Flood-It	cpc-trees		_	NP-hard	Corol. 4.1.14
	trees	$c \ge 3$	_	NP-hard	Coroll. 4.1.15
	trees	$d \ge 4$	_	NP-hard	Corol. 4.1.13
	pc-trees		k	W[1]-hard	Corol. 4.1.16
	cpc-trees		r	FPT	Coroll. 4.1.18
	graphs	r	_	Polynomial	Theor. 4.1.12
	trees		o, k	W[1]-hard	Corol. 4.1.16
	pc-trees		k, r	FPT	Corol. 4.1.17

Many complexity issues on Flood-It and Free-Flood-It have recently been investigated, and these games have presented interesting behavior when are played on non-grid graphs.

Analyzing the computational complexity of these games on classes of graphs shuch as power of paths, power of cycles, circular grids and graphs with bounded vertex cover, we conclude that: (i) Flood-it can be solved in polynomial time on P_n^2 , C_n^2 , $2 \times n$ circular grids; (ii) Free-Flood-it is NP-hard on P_n^2 , C_n^2 and $2 \times n$ circular grids.

On the parameterized complexity point of view, we conclude that: Flood-it game played on graphs with bounded minimum vertex cover is fixed-parameter tractable. In addition, we show a polynomial kernelization algorithm for Flood-it when besides the minimum vertex cover, it is fixed the number of colors as parameter.

4.4.1 Open Problems

In Subsection 4.1.3 it was shown, using Corollary 4.1.7 or Corollary 4.1.8, that Weighted-Flood-It and Flood-It on cpc-trees are fixed-parameter tractable when considering the number of bad moves (r) as the parameter. We present two open problems on flood-filling games:

- Does Flood-It(r) remain fixed-parameter tractable on general pc-trees?
- Does Weighted-Flood-It(r) remain fixed-parameter tractable on general pc-trees?

Both questions seem to be reasonable conjectures, but in [35] it was shown within the scope of Shortest Common Supersequence that if the analog version of Flood-It(r) on pc-trees (denoted BOUNDED DUPLICATION SCS III) is fixed-parameter tractable then DIRECTED FEEDBACK VERTEX SET is also in FPT, which is apparently unlikely. [31].

Chapter 5 On P₃-convexity

"The only place success comes before work is in the dictionary."

Vince Lombardi

In this chapter, we study new complexity aspects of P_3 -convexity restricted to graphs with bounded maximum degree. More specifically we are interested in identifying either the minimum P_3 -geodetic set or the minimum P_3 -hull set of such graphs, from which the whole vertex set of G is obtained either after one or eventual iterations, respectively. Each iteration adds to a set S all vertices of $V(G) \setminus S$ with two neighbors in S. We prove that: (i) the minimum P_3 -hull set of a graph G can be found in polynomial time when $\delta(G) \geq \frac{n(G)}{c}$ (for some constant c); (ii) determining the size of the minimum P_3 -hull set of a graph remains NP-hard even on planar graphs with maximum degree four; (iii) the minimum P_3 -hull set of a cubic graph can be found in polynomial time; (iv) the minimum P_3 -hull set can be found in polynomial time in graphs with minimum feedback vertex set of bounded size and no vertex of degree two; (v) it is NP-hard to determine the size of the minimum P_3 -geodetic set of a planar graph with maximum degree three.

Let G = (V, E) be a graph. For $U \subseteq V$, let the interval I[U] of U in G be the set $U \cup \{u \in V(G) \setminus U \mid |N_G(u) \cap U| \geq 2\}$. A set S of vertices of G is P_3 -geodetic if I[S] contains all vertices of G. The P_3 -geodetic number $g_{P_3}(G)$ of a graph G is defined as the minimum cardinality of a P_3 -geodetic set. The decision problem related to determining the P_3 -geodetic number is known to be NP-complete for general graphs, and coincides with the well-studied 2-domination number [51, 49, 29, 50, 23].

A P_3 -hull set U of G is a set of vertices such that:

• $U^0 = U$

- $U^k = I[U^{k-1}], \text{ for } k \ge 1.$
- $\exists k \ge 0 \mid U^k = V(G)$

We define $H_G(S) \subseteq V(G)$ as $I[S]^{k+1}$ such that $I[S]^{k+1} = I[S]^k$, $k \ge 0$. The cardinality of a minimum P_3 -hull set of G is the P_3 -hull number of G, denoted by $h_{p3}(G)$. Again, the decision problem related to determining the P_3 -hull number of a graph is still a well known NP-complete problem [21].

According to [22], as one of the most elementary models of the spreading of a property within a network – like sharing an idea or disseminating a virus – one can consider a graph G, a set U of vertices of G that initially possesses the property, and an iterative process whereby new vertices u are added to U whenever sufficiently many neighbors of u are already in U. The simplest choice leads to the *irreversible 2-threshold processes* by Dreyer and Roberts [54]. Similar models were studied in various contexts, such as statistical physics, social networks, marketing, and distributed computing under different names such as bootstrap percolation, influence dynamics, local majority processes, irreversible dynamic monopolies, catastrophic fault patterns, and many others [4, 52, 8, 21, 22, 54].

In the next sections, we analyze the complexity of these problems when some parameters related to the maximum and minimum degree of a graph are known. In the following subsection we review some results on planar satisfiability problems. In Section 2 we present some results on finding a minimum P_3 -hull set of graphs with bounded degree. Finally, in Section 3 we analyze complexity aspects of finding a minimum P_3 -geodetic set on planar graphs with bounded degree.

5.0.2 PLANAR SAT-AM3

SAT-AM3 [42]

Instance: A set $F = \{C_1, C_2, \ldots, C_m\}$ of clauses, built on a finite set $X = \{x_1, x_2, \ldots, x_n\}$ of boolean variables, such that each clause contains at most three literals, each variable appears at most three times, and each literal occurs at most twice.

Question: Is there a truth assignment to the variables in X that satisfies F?

SAT-AM3 is an NP-complete problem [42]. In [42] the problem was not defined with the restriction of each literal occurs at most twice, but without loss of generality, if a literal l occurs three times, the clauses containing l can be considered satisfied and removed from the formula F to be analyzed. Another variant of SAT is described below.

PLANAR 3-SAT [42] **Instance:** A set $F = \{C_1, C_2, \ldots, C_m\}$ of clauses, built on a finite set $X = \{x_1, x_2, \ldots, x_n\}$ of boolean variables, where each clause contains at most three literals, and the bipartite graph $H_F = (V, E)$ such that $V = \{w_{c_1}, w_{c_2}, \ldots, w_{c_m}\} \cup \{v_{x_1}, v_{x_2}, \ldots, v_{x_n}\}$ and E contains exactly those pairs (w_{c_i}, v_{x_j}) such that either x_j or $\neg x_j$ belongs to the clause C_i , is planar.

Question: Is there a truth assignment to the variables in X that satisfies F?

Note that not every instance of SAT-AM3 is an instance of PLANAR 3-SAT. For example, $F = (\neg x_1 + x_2 + x_3)(x_2 + \neg x_3 + \neg x_5)(x_1 + \neg x_2 + x_4)(x_3 + \neg x_4)(\neg x_1 + x_5)$ is non-planar. However, it is well known [42, 61] that PLANAR 3-SAT is also an NP-complete problem.

At this point, we describe an intersection of these problems.

PLANAR SAT-AM3

Instance: A set $F = \{C_1, C_2, \ldots, C_m\}$ of clauses, built on a finite set $X = \{x_1, x_2, \ldots, x_n\}$ of boolean variables, where each clause contains at most three literals, each variable appears at most three times, each literal occurs at most twice, and the bipartite graph $H_F = (V, E)$ such that $V = \{w_{c_1}, w_{c_2}, \ldots, w_{c_m}\} \cup \{v_{x_1}, v_{x_2}, \ldots, v_{x_n}\}$ and E contains exactly those pairs (w_{c_i}, v_{x_j}) such that either x_j or $\neg x_j$ belongs to the clause C_i , is planar.

Question: Is there a truth assignment to the variables in X that satisfies F?

Lemma 5.0.1. PLANAR SAT-AM3 is NP-complete.

Proof. It is easy to see that the problem is in NP. To prove the hardness, we perform a reduction from PLANAR 3-SAT. Consider a general Planar 3-SAT expression F in which x_i appears k_i times. Assign F' = F, and for each x_i in F' replace the first occurrence of x_i by x_i^1 , the second by x_i^2 , and so on, where $x_i^1, x_i^2, \ldots, x_i^{k_i}$ are new variables. Add $(\neg x_i^1, x_i^2), (\neg x_i^2, x_i^3), \ldots, (\neg x_i^{k_i}, x_i^1)$ to F'. Clearly, F' is satisfiable if and only if F is satisfiable.

By Kuratowski's theorem a finite graph is planar if and only if it does not contain a subgraph that is a subdivision of K_5 or $K_{3,3}$. To show that $H_{F'}$ is planar, just observe that if $H_{F'}$ contains a subgraph that is a subdivision of K_5 or $K_{3,3}$, we can construct a subgraph of H_F that is a subdivision of K_5 or $K_{3,3}$ replacing paths composed of new variables and clauses by original edges, which gives us a contradiction.

5.1 P_3 -Hull Set

In this section we consider both search and decision problems on P_3 -hull sets.

 P_3 -Hull Set

Instance: A graph G.

Goal: Find a P_3 -hull set of G with minimum cardinality.

 P_3 -Hull Number

Instance: A graph G; an integer k.

Goal: Decide if G has a P_3 -hull set with cardinality at most k.

Note that P_3 -HULL NUMBER is clearly in NP. Moreover, it is easy to see that if P_3 -HULL NUMBER is NP-complete then P_3 -HULL SET is NP-hard.

Let n(G) be the number of vertices of G, $N_G(x)$ the neighborhood of a vertex x in G, $d_G(x) = |N_G(x)|$ the degree of vertex x in G, $\delta(G)$ and $\Delta(G)$ the minimum and maximum degree of a vertex in G, respectively, and $U \in \binom{V(G)}{k}$ a subset of V(G) such that |U| = k.

Lemma 5.1.1. Let k be a positive integer. If G is a graph, then

$$\Delta_k(G) := \max\left\{ \left| \bigcap_{x \in U} N_G(x) \right| : U \in \binom{V(G)}{k} \right\} \ge n(G) \frac{\binom{\delta(G)}{k}}{\binom{n(G)}{k}}.$$

Proof. Let $\mathcal{R} = \left\{ (u, U) : u \in V(G), U \in \binom{V(G)}{k}, u \in \bigcap_{x \in U} N_G(x) \right\}$. Since for every vertex v of G there are $\binom{d_G(v)}{k} \ge \binom{\delta(G)}{k}$ pairs (u, U) in \mathcal{R} with u = v, we have $|\mathcal{R}| \ge n(G)\binom{\delta(G)}{k}$. Conversely, by the definition of $\Delta_k(G)$, for every set $V \in \binom{V(G)}{k}$, there are at most $\Delta_k(G)$ pairs (u, U) in \mathcal{R} with U = V, which implies $|\mathcal{R}| \le \Delta_k(G)\binom{n(G)}{k}$. \Box

Theorem 5.1.1. Let c be a positive integer. If G is a graph with $\delta(G) \geq \frac{n(G)}{c}$, then

$$h_{P_3}(G) \le 2\left[\frac{\log(2c)}{\log\left(\frac{2c^2}{2c^2-1}\right)}\right] + 2c^3.$$

Proof. In order to construct a small P_3 -hull set of G we describe an inductive construction of a sequence G_1, \ldots, G_k of induced subgraphs of G such that

- $G_i = G H_G(S_{i-1})$ for a set S_{i-1} of at most 2(i-1) vertices of G,
- $n(G_i) \le n(G) \left(1 \frac{1}{2c^2}\right)^{i-1}$, and

•
$$\delta(G_i) \ge \frac{n(G_i)}{c}$$

for $i \in [k]$.

Let $G_1 = G$ and $S_0 = \emptyset$.

Now let *i* be such that G_i and S_{i-1} are defined. If G_i is complete or $n(G_i) < 2c^3$, then terminate the construction of the sequence and set *k* to *i*. Since

$$h_{P_3}(G) \le |S_{k-1}| + h_{P_3}(G_k) \le 2(k-1) + 2c^3,$$

it suffices to bound k in order to complete the proof.

Therefore, we may assume that G_i is not complete and that $n(G_i) \ge 2c^3$. By Lemma 5.1.1, there are two vertices u_i and v_i of G_i with at least

$$n(G_i)\frac{\binom{\delta(G_i)}{2}}{\binom{n(G_i)}{2}} \ge n(G_i)\frac{\binom{n(G_i)}{2}}{\binom{n(G_i)}{2}} = \frac{n(G_i)(n(G_i) - c)}{c^2(n(G_i) - 1)} \ge \frac{n(G_i)}{2c^2}$$

common neighbors. Let $S_i = S_{i-1} \cup \{u_i, v_i\}$ and $G_{i+1} = G - H_G(S_i)$. We obtain

$$\begin{split} n(G_{i+1}) &= n(G) - |H_G(S_i)| \\ &\leq n(G) - |H_G(S_{i-1}) \cup H_{G_i}(\{u_i, v_i\})| \\ &\leq n(G) - |H_G(S_{i-1})| - |H_{G_i}(\{u_i, v_i\})| \\ &= n(G_i) - |H_{G_i}(\{u_i, v_i\})| \\ &\leq n(G_i) - \frac{n(G_i)}{2c^2} \\ &= n(G_i) \left(1 - \frac{1}{2c^2}\right) \\ &\leq n(G) \left(1 - \frac{1}{2c^2}\right)^i. \end{split}$$

Since $G_{i+1} = G - H_G(S_i)$, we have $\delta(G_{i+1}) \ge \delta(G) - 1 \ge \frac{n(G)}{c} - 1$. Therefore,

$$\frac{\delta(G_{i+1})}{n(G_{i+1})} \ge \frac{\frac{n(G)}{c} - 1}{n(G_i)\left(1 - \frac{1}{2c^2}\right)} \ge \frac{\frac{n(G_i)}{c} - 1}{n(G_i)\left(1 - \frac{1}{2c^2}\right)} \ge \frac{1}{c}.$$

Since the minimum degree of all graphs G_i in the sequence is at least $\delta - 1$, the value of k is less than or equal to the smallest integer r with

$$n(G)\left(1-\frac{1}{2c^2}\right)^{r-1} \le \frac{n(G)}{c} - 1.$$

Since $\frac{n(G)}{c} - 1 \ge \frac{n(G)}{2c}$, we obtain

$$k \le \left\lceil \frac{\log(2c)}{\log\left(\frac{2c^2}{2c^2-1}\right)} \right\rceil + 1,$$

which completes the proof.

Corollary 5.1.1. A minimum P_3 -hull set of a graph G with $\delta(G) \geq \frac{n(G)}{c}$ (for some constant c) can be found in polynomial time.

Proof. The proof follows immediately from Theorem 5.1.1.

Theorem 5.1.2. P_3 -HULL NUMBER remains NP-complete on planar graphs with maximum degree four.

Proof. To prove that deciding whether the P_3 -hull number of a graph G is less than or equal to k is NP-complete, we perform a reduction from PLANAR SAT-AM3, proved to be NP-complete in Lemma 5.0.1. Here cross edges are meant in the usual sense of a planar graph: edges crossing other edges in a specific embedding of a graph in the plane.

Given an instance F of PLANAR SAT-AM3 we construct an instance G of P_3 -HULL SET as follows:

• For each variable x_i of F, create a gadget G_{x_i} composed of 62 vertices as illustrated in Figure 5.1. Note that G_{x_i} is composed of two subgadgets g_{x_i} and $g_{\bar{x}_i}$ which represent the literals x_i and \bar{x}_i respectively.



Figure 5.1: Gadget G_{x_i} .

- For each clause C_j of F, create a gadget G_{c_j} composed of the cycle $b_{c_j}^1$, $b_{c_j}^2$, $b_{c_j}^3$, $b_{c_j}^4$, $b_{c_j}^5$, $b_{c_j}^6$, $b_{c_j}^7$, $b_{c_j}^8$ plus the vertices $b_{c_j}^9$, $b_{c_j}^{10}$, $b_{c_j}^{11}$, $b_{c_j}^{12}$, $b_{c_j}^{13}$, $b_{c_j}^{14}$, $b_{c_j}^{15}$, $b_{c_j}^{16}$ and edges $(b_{c_j}^1, b_{c_j}^9), (b_{c_j}^2, b^{10}_{c_j}), (b_{c_j}^3, b^{11}_{c_j}), (b_{c_j}^4, b_{c_j}^{12}), (b_{c_j}^5, b_{c_j}^{13}), (b_{c_j}^6, b_{c_j}^{14}), (b_{c_j}^7, b_{c_j}^{15}), (b_{c_j}^8, b_{c_j}^{16})$. Figure 5.2 illustrates a gadget G_{c_i} .
- If the literal x_i occurs twice in F then create the vertices $f_{x_i}^1$, $f_{x_i}^2$, and add edges $(f_{x_i}^1, a_{x_i}^7), (f_{x_i}^2, a_{x_i}^8)$. Otherwise, create only $f_{x_i}^1$ and add $(f_{x_i}^1, a_{x_i}^7)$.



Figure 5.2: Gadget G_{c_i} .

- If the literal \bar{x}_i occurs twice in F then create the vertices $f_{\bar{x}_i}^1$, $f_{\bar{x}_i}^2$, and add edges $(f_{\bar{x}_i}^1, a_{\bar{x}_i}^7), (f_{\bar{x}_i}^2, a_{\bar{x}_i}^8)$. Otherwise, create only $f_{\bar{x}_i}^1$ and add $(f_{\bar{x}_i}^1, a_{\bar{x}_i}^7)$.
- For each clause C_j do:
 - 1. if x_i is the first literal of C_j then: if C_j contains the first occurrence of x_i then add edges $(a_{x_i}^7, b_{c_j}^1), (a_{x_i}^9, b_{c_j}^2)$; else add edges $(a_{x_i}^{10}, b_{c_j}^1), (a_{x_i}^8, b_{c_j}^2)$.
 - 2. if x_i is the second literal of C_j then: if C_j contains the first occurrence of x_i then add edges $(a_{x_i}^7, b_{c_j}^5), (a_{x_i}^9, b_{c_j}^6)$; else add edges $(a_{x_i}^{10}, b_{c_j}^5), (a_{x_i}^8, b_{c_j}^6)$.
 - 3. if x_i is the third literal of C_j then: if C_j contains the first occurrence of x_i then add edges $(a_{x_i}^7, b_{c_j}^7), (a_{x_i}^9, b_{c_j}^8)$; else add edges $(a_{x_i}^{10}, b_{c_j}^7), (a_{x_i}^8, b_{c_j}^8)$. If this step generates cross edges, remove the newly created edges, and repeat this step replacing $b_{c_j}^7$ and $b_{c_j}^8$ by $b_{c_j}^3$ and $b_{c_j}^4$, respectively. This operation keeps the graph planar, as one can check by verifying all possible configurations.
 - 4. if \bar{x}_i is the first literal of C_j then: if C_j contains the first occurrence of \bar{x}_i then add edges $(a_{\bar{x}_i}^7, b_{c_i}^2), (a_{\bar{x}_i}^9, b_{c_i}^1)$; else add edges $(a_{\bar{x}_i}^{10}, b_{c_i}^2), (a_{\bar{x}_i}^8, b_{c_i}^1)$.
 - 5. if \bar{x}_i is the second literal of C_j then: if C_j contains the first occurrence of \bar{x}_i then add edges $(a_{\bar{x}_i}^7, b_{c_j}^6), (a_{\bar{x}_i}^9, b_{c_j}^5)$; else add edges $(a_{\bar{x}_i}^{10}, b_{c_j}^6), (a_{\bar{x}_i}^8, b_{c_j}^5)$.
 - 6. if \bar{x}_i is the third literal of C_j then: if C_j contains the first occurrence of \bar{x}_i then add edges $(a_{\bar{x}_i}^7, b_{c_j}^8), (a_{\bar{x}_i}^9, b_{c_j}^7)$; else add edges $(a_{\bar{x}_i}^{10}, b_{c_j}^8), (a_{\bar{x}_i}^8, b_{c_j}^7)$. If this step generates cross edges, remove the newly created edges, and repeat this step replacing $b_{c_j}^7$ and $b_{c_j}^8$ by $b_{c_j}^3$ and $b_{c_j}^4$, respectively. As above, this operation keeps the graph planar, as one can check by verifying all possible configurations.

Figure 5.3 shows a graph constructed from $F = (x_1)(x_2)(x_1 + \bar{x_2} + x_3)(\bar{x_1} + \bar{x_2} + \bar{x_3})(\bar{x_3})$.



Figure 5.3: Graph obtained from $F = (x_1)(x_2)(x_1 + \bar{x_2} + x_3)(\bar{x_1} + \bar{x_2} + \bar{x_3})(\bar{x_3})$.

Let G be the graph obtained by the construction above from an instance F of PLANAR SAT-AM3. At this point, we will prove that F is satisfiable if and only if G has a hull set of size 8m + 23n, where m is the number of clauses, and n is the number of variables of F.

If F is satisfiable then we can obtain a P_3 -hull set S of G by first adding all the pendant vertices of G to S. Note that G has 8m + 22n pendant vertices. Let A be a truth assignment of F. If $x_i = true$ in A we add $a_{x_i}^2$ to S, else we add $a_{\bar{x}_i}^2$ to S. As A is a truth assignment of F, each gadget G_{c_j} will be contaminated and consequently all vertices of G will be contaminated. Hence S is a P_3 -hull set of size 8m + 23n.

Conversely, if G has a P_3 -hull set S of size 8m + 23n, S contains 8m + 22n pendant vertices and n non-pendant vertices of G. As we can observe in each gadget G_{x_i} of G, there is a subgraph B_{x_i} such that every vertex v of B_{x_i} is not a pendant vertex and either it is adjacent to only one leaf and has no non-pendant neighbor outside B_{x_i} , or v has only one neighbor outside B_{x_i} . Figure 5.4 illustrates a gadget G_{x_i} and its subgraph B_{x_i} . Consequently, each subgraph B_{x_i} must have exactly one vertex in S, otherwise either S is not a P_3 -hull set or S has size greater than 8m + 23n. At this point we can construct an assignment A of F by setting $x_i = true$ if and only if $S \cap V(g_{x_i}) \cap B_{x_i} \neq \emptyset$. By construction, we can see that A is a truth assignment of F.



Figure 5.4: Gadget G_{x_i} and its subgraph B_{x_i} inside the rectangle. The white vertices are pendant vertices in G and are not contained in B_{x_i} .

Lemma 5.1.2. Let G be a cubic graph. $S \subseteq V(G)$ is a P_3 -hull set of G if and only if S is also a feedback vertex set of G.

Proof. Let G be a graph with maximum degree three and S be a P_3 -hull set of G. If $G[V \setminus S]$ has a cycle C then each vertex $v \in C$ has at most one neighbor outside C, and consequently C is not in the hull of S, which is a contradiction because S is a P_3 -hull set of G.

Conversely, let B be a feedback vertex set of G. As $G[V \setminus B]$ is a forest and G is cubic, all pendant vertices of $G[V \setminus B]$ are in $H_G(B)$; by removing these pendant vertices of $G[V \setminus B]$ we obtain a forest T where each leaf of T has two neighbors in $H_G(B)$. Applying this step recursively we can see that all vertices of $G[V \setminus B]$ are in $H_G(B)$.

Lemma 5.1.3. [90] A minimum feedback vertex set of a graph G with maximum degree at most three can be found in polynomial time.

Corollary 5.1.2. A minimum P_3 -hull set of a cubic graph can be found in polynomial time.

Proof. The proof follows immediately from Lemma 5.1.2 and Lemma 5.1.3.

Theorem 5.1.3. Let \mathscr{F} be the class of graphs with no vertex of degree two and with minimum feedback vertex set of size bounded by a constant c. Then P_3 -HULL SET on \mathscr{F} can be solved in polynomial time. **Proof.** Let $G \in \mathscr{F}$. As G has minimum feedback vertex set of size bounded by a constant c, we can find a minimum feedback vertex set B of G in polynomial time. Let L be the set of pendant vertices in G, and let $T = G \setminus \{B \cup L\}$. Since G has no vertex of degree two, each leaf of T has at least two neighbors in $\{B \cup L\}$ and just as in the proof of Lemma 5.1.2, $\{B \cup L\}$ is a hull set of G. As L is in any hull set of G, it is sufficient to examine all subset of vertices in $V(G) \setminus L$ of size at most c to find a minimum P_3 -hull set of G.

Corollary 5.1.3. P_3 -HULL NUMBER remains NP-complete on planar graphs with maximum degree three.

Proof. In Theorem 5.1.2 we show that P_3 -HULL SET remains NP-complete on planar graphs with maximum degree four.

Let G be a graph with n vertices, maximum degree four, and containing ℓ vertices with degree four. We construct a graph G' with maximum degree three and $n + 18\ell$ vertices such that G has a P_3 -hull set of size k if and only if G' has a P_3 -hull set of size $k + 5\ell$. The construction is described below.

- set G' = G;
- for each vertex $u_i \in V(G)$, create in G' a gadget g_{u_i} composed of a cycle $v_{u_i}^1, v_{u_i}^2, v_{u_i}^3, v_{u_i}^4, \dots, v_{u_i}^{18}$ (formed by new vertices), plus the edges $(v_{u_i}^1, v_{u_i}^3), (v_{u_i}^5, v_{u_i}^7), (v_{u_i}^{10}, v_{u_i}^{12}), (v_{u_i}^{14}, v_{u_i}^{16}), (v_{u_i}^8, v_{u_i}^{18}), (v_{u_i}^9, v_{u_i}^17).$
- let $u_j, u_{j+1}, u_{j+2}, u_{j+3}$ be the neighbours of u_i in G; add to G' the edges $(u_j, v_{u_i}^2), (u_{j+1}, v_{u_i}^6), (u_{j+2}, v_{u_i}^1 1), (u_{j+3}, v_{u_i}^1 5)$, and remove u_i .

Note that G' is a graph with maximum degree three. As G' has maximum degree three, for each cycle inside G' at least one vertex must be in any P_3 -hull set of G', which implies that any gadget g_{u_i} of G' has at least five vertices in any P_3 -hull set of G'.

As illustrated in Figure 5.5, six internal vertices are enough to contaminate a gadget g_{u_i} of G'. Note that for a gadget g_{u_i} in G', even when one vertex of the subset $\{u_j, u_{j+1}, u_{j+2}, u_{j+3}\}$ is contaminated, it is still necessary to use six internal vertices to contaminate the gadget g_{u_i} . On the other hand, if any two vertices of the subset $\{u_j, u_{j+1}, u_{j+2}, u_{j+3}\}$ are contaminated then it is necessary to use only five internal vertices to contaminate the gadget g_{u_i} . See Figure 5.6.

Thus, if there is a P_3 -hull set D of G with size k then there is a P_3 -hull set D' of G'with size $k + 5\ell$ using the vertices in $D \cap V(G')$ plus 5 vertices for each gadget g_{u_i} such that $u_i \notin D$ and 6 vertices for each gadget g_{u_i} such that $u_i \in D$. Conversely, if G' has a P_3 -hull set D' of size $k + 5\ell$ then we construct a P_3 -hull set D of G of size at most k using the vertices $D' \cap V(G)$ and adding u_i if and only if $V(g_{u_i}) \cap D' \ge 6$ (for every i).



Figure 5.5: Six internal vertices to contaminate a gadget g_{u_i} .



Figure 5.6: Two vertices in $\{u_j, u_{j+1}, u_{j+2}, u_{j+3}\}$ are contaminated implying to use only five internal vertices to contaminate the gadget g_{u_i} .

5.2 P_3 -Geodetic Set

Now we consider the following decision problem:

 P_3 -Geodetic Number

Instance: A graph G; an integer k.

Goal: Decide if G has a P_3 -geodetic set with cardinality at most k.

Note that P_3 -GEODETIC NUMBER is clearly in NP.

As DOMINATING SET is NP-complete even restricted to planar graphs with maximum degree three [42], it is easy to see that P_3 -GEODETIC NUMBER problem remains NPcomplete on planar graphs with maximum degree four. Just take an instance G of such restricted DOMINATING SET problem and construct a graph G' by adding a new vertex w_v and a new edge (v, w_v) for each vertex v of G. Note that G has a dominating set of size k if and only if G' has a P_3 -geodetic set of size n + k. As G is a planar graph with maximum degree 3, G' is a planar graph with maximum degree 4.

As P_3 -GEODETIC NUMBER is NP-complete on planar graphs with maximum degree four, and trivially solvable in polynomial time on graphs with maximum degree two, it is natural to ask about the complexity of P_3 -Geodetic Set on planar graphs with maximum degree 3.

Theorem 5.2.1. P_3 -GEODETIC NUMBER remains NP-complete on planar graphs with maximum degree three.

Proof. Deciding whether the P_3 -geodetic number of a graph G is less than or equal to k is clearly a problem in NP. To prove the NP-hardness we perform a reduction from PLANAR SAT-AM3, proved to be NP-complete in Lemma 5.0.1. Given an instance F of PLANAR SAT-AM3 we construct an instance G of P_3 -GEODETIC SET as follows:

- for each variable x_i do: create in G a gadget g_{x_i} composed of a cycle $f_{x_i}^1, t_{x_i}^1, a_{x_i}^2, f_{x_i}^2, -t_{x_i}^2, a_{x_i}^3, a_{x_i}^4, a_{x_i}^3, a_{x_i}^4, f_{x_i}^2, a_{x_i}^3, a_{x_i}^4$;
- for each clause C_i containing at most two literals do: create in G a gadget g_{c_i} composed of the vertices c_i^1 , c_i^2 and edge (c_i^1, c_i^2) ;
- for each clause C_j containing exactly three literals do: create in G a gadget g_{c_j} composed of the vertices $c_j^1, c_j^2, c_j^3, l_j^1, l_j^2$ and the edges $(c_j^1, c_j^2), (c_j^1, c_j^3), (c_j^1, l_j^1), (c_j^3, l_j^2);$
- for each clause C_j of F do:
 - 1. add an edge $(c_j^2, t_{x_i}^p)$ if x_i is the first or second literal of C_j and it is the *p*-th occurrency of x_i $(1 \le p \le 2)$;
 - 2. add an edge $(c_j^2, f_{x_i}^p)$ if $\neg x_i$ is the first or second literal of C_j and it is the *p*-th occurrency of $\neg x_i$ $(1 \le p \le 2)$;

- 3. add an edge $(c_j^3, t_{x_i}^p)$ if x_i is the third literal of C_j and it is the *p*-th occurrency of x_i $(1 \le p \le 2)$;
- 4. add an edge $(c_j^3, f_{x_i}^p)$ if $\neg x_i$ is the third literal of C_j and it is the *p*-th occurrency of $\neg x_i$ $(1 \le p \le 2)$.

At this point, we show that given an instance F of SAT-AM3, where n is the number of variables, m_1 the number of clauses with at most two literals, and m_2 the number of clauses with three literals, by the construction above we obtain a graph G such that: Fis satisfiable if and only if G has a P_3 -geodetic set S of size k, where $k = 4n + m_1 + 3m_2$.



Figure 5.7: (a) - (d) Choices of vertices in S_A that imply in at least 5 vertices to be added to S_A ; thicker edges mean that one of its endpoints must be added to S_A ; (e) - (h) Choices of vertices in S_A that imply in exactly 4 vertices to be added to S_A .

Let F be a satisfiable formula and A be a truth assignment of F. We obtain a P_3 geodetic set S_A of G from A as follows: (i) every vertex with degree one is added to S_A ;
(ii) if $x_i = true$ in A then $t_{x_i}^1, t_{x_i}^2, a_{x_i}^2, a_{x_i}^4$ are added to S_A ; (iii) if $x_i = false$ in A then $f_{x_i}^1, f_{x_i}^2, a_{x_i}^1, a_{x_i}^3$ are added to S_A ; (iv) for each clause C_i with three literals, if c_i^3 has two
neighbors in S_A then c_i^2 is added to S_A , otherwise c_i^1 is added to S_A . As A is a truth
assignment of F, each gadget g_{c_i} of G has at least one neighbor in $S_A \cap \{\bigcup_{i=1}^{n} V(g_{x_i})\}$;
consequently, S_A is a P_3 -geodetic set of G of size $k = 4n + m_1 + 3m_2$.

Conversely, Let S_A be a P_3 -geodetic set of G of size $k = 4n + m_1 + 3m_2$. We construct a truth assignment A for the variables x_1, x_2, \ldots, x_n that satisfies all the clauses in F as follows. Any P_3 -geodetic set of G contains: (i) at least one vertex of each gadget g_{c_i} if C_i has at most two literals; (ii) at least three vertices of each gadget g_{c_i} if C_i has three literals; (iii) at least four vertices of each gadget g_{x_i} . As S_A has size k, each gadget g_{x_i} has exactly four vertices in S_A , and at most two of these vertices has degree three in G: either $t_{x_i}^1$ and $t_{x_i}^2$, or $f_{x_i}^1$ and $f_{x_i}^2$. See Figure 5.7. At this point, we can construct a truth assignment A of F by assigning $x_i = true$ if and only if $t_{x_i}^1 \in S_A$ or $t_{x_i}^2 \in S_A$ and $t_{x_i}^2$ has degree three in G. By (i) and (ii), each gadget g_{c_i} must have at least one neighbor in S_A , otherwise either S_A would not be a P_3 -geodetic set or we would have $|S_A| > k$. Consequently, by the construction of G and A, if S_A is a P_3 -geodetic set of G of size k then A is a truth assignment of F.

Figure 5.8 illustrates a boolean formula F and the graph G obtained from F by the construction above. A possible P_3 -geodetic set S_A is colored red.



Figure 5.8: (a) Satisfiable boolean formula $F = (x_1)(x_2)(x_1 + \neg x_2)(\neg x_1 + \neg x_2 + \neg x_3)(\neg x_3);$ (b) Graph G constructed from F.

It is easy to see that G has maximum degree three. To show that G is planar, we can split G in two subgraphs $G_x = \{\bigcup_{i=1}^{n} g_{x_i}\}$ and $G_c = \{\bigcup_{i=1}^{m} g_{c_i}\}$. Note that G_x and G_c are both planar graphs. By contracting each graph g_{x_i} and each gadget g_{c_j} of G into a single vertex, we obtain the bipartite graph H_F which by assumption is planar. Hence, G is also a planar graph.

5.2.1 Parameters

Now, we will analyze the complexity of finding a P_3 -geodetic set of size at most k, where k is fixed as parameter. For short, we denote this parameterized version by P_3 -GEODETIC SET(k).

Lemma 5.2.1. P_3 -GEODETIC SET(k) is W/2-hard.

Proof. We use a reduction from DOMINATING SET(k'), a well known W[2]-complete problem [31]. Given an instance (G', k') of DOMINATING SET(k') we construct an instance (G, k) of P_3 -GEODETIC SET(k) as follows.

- let H be a K_3 -graph where V(H) = a, b, c;
- set $G = G' \cup H$;
- add in G an edge from a to each vertex in $V(G) \cap V(G')$;
- assign k = k' + 2.

At this point, it is easy to see that G' has a dominating set of size k' if and only if G has a P_3 -geodetic set of size k, where k = k' + 2.

Given that P_3 -GEODETIC SET remains NP-complete even on planar graphs with maximum degree three (Theorem 5.2.1) and P_3 -GEODETIC SET(k) is W[2]-hard (Lemma 5.2.1). It is natural to ask about the complexity of finding in a graph G a P_3 -geodetic set of size at most k, where k and the maximum degree of G are fixed as parameters. For short, we denote this parameterized version by P_3 -GEODETIC SET (k, Δ) .

Lemma 5.2.2. P_3 -GEODETIC SET (k, Δ) is fixed-parameter tractable.

Proof. Let G be an instance of P_3 -GEODETIC SET (k, Δ) . As G has maximum degree bounded by Δ , if G has a P_3 -geodetic set S of size k then $|V \setminus S| \leq k.\Delta$ because S dominates only $k.\Delta$ vertices. Thus, if $|V(G)| \leq k + k.\Delta$ then G can be exhaustively analysed in FPT-time, otherwise G does not have a P_3 -geodetic set S of size k.

5.3 Open Problem

In this chapter we analyze the complexity of problems related to convexity P_3 of graphs. The results show that the maximum degree is not a source of polynomial time intractability for P_3 -GEODETIC NUMBER and P_3 -HULL NUMBER. Thus we have the following question.

• Which sets of parameters are sources of polynomial time intractability of the problems related to convexity P_3 ? In other words, for which parameters the problems become fixed-parameter tractable?

Chapter 6

Induced Matchings close to Maximum Matchings

"Success consists of going from failure to failure without loss of enthusiasm."

Winston Churchill

Extending results of Kobler and Rotics (Finding maximum induced matchings in subclasses of claw-free and P_5 -free graphs, and in graphs with matching and induced matching of equal maximum size, Algorithmica 37 (2003) 327-346), Cameron and Walker (The graphs with maximum induced matching and maximum matching the same size, Discrete Math. 299 (2005) 49-55) gave a complete structural description of the graphs G where the matching number $\nu(G)$ equals the induced matching number $\nu_2(G)$. We present a short proof of their result and use it to study graphs G with $\nu(G) - \nu_2(G) \leq k$. We show that the recognition of these graphs can be done in polynomial time for fixed k, and is fixed parameter tractable when parameterized by k for graphs of bounded maximum degree.

We consider finite, simple, and undirected graphs, and use standard terminology and notation [62]. For a graph G, the vertex set and the edge set are denote by V(G) and E(G), respectively. For two sets X and Y of vertices of G, the *distance* between X and Yin G is the minimum number of edges of a path in G between a vertex in X and a vertex in Y. Considering an edge as a set containing two distinct vertices, this defines distances of edges. A *matching* of a graph G is a set M of edges of G at pairwise distance at least 1. The *matching number* $\nu(G)$ of G is the maximum cardinality of a matching in G, and a matching with $\nu(G)$ edges is a *maximum matching* of G. Matchings in graphs are among the most classical topics in graph theory [62]. Stockmeyer and Vazirani [89] introduced the notion of an *induced matching* as a set M of edges of G at pairwise distance at least 2. Equivalently, a matching M is induced if the graph (V(G), M) has maximum degree at most 1. The *induced matching number* $\nu_2(G)$ of G is the maximum cardinality of an induced matching in G, and an induced matching with $\nu_2(G)$ edges is a *maximum induced matching*. More generally, if k is a positive integer, then a set M of edges of a graph Gis a *k*-matching of G if the edges in M have pairwise distance at least k. The *k*-matching *number* $\nu_k(G)$ of G is the maximum cardinality of a *k*-matching in G, and a *k*-matching with $\nu_k(G)$ edges is a *maximum k*-matching.

While maximum matchings can be found efficiently [62], it is algorithmically hard to find a maximum induced matching [89, 16]. It is even hard to approximate the induced matching number under substantial restrictions, and efficient exact and approximation algorithms have been proposed for several special graph classes (cf. [28, 73] for a detailed discussion). The fixed parameter tractability [75] of induced matchings when parameterized by their cardinality was studied in [74, 73, 28]. While this problem is W[1]-hard in general, it was shown to be fixed parameter tractable for several restricted graph classes.

In the present paper we study graphs where the matching number is not much larger than the induced matching number. Kobler and Rotics [57] showed that the graphs where these two numbers coincide, can be recognized efficiently. Cameron and Walker [19] extended this result and gave a complete structural description of these graphs. In Section 6.1 we review the results from [57, 19] and present shorter proofs. In Section 6.2 we study graphs G where $\nu(G) - \nu_2(G) \leq k$. We show that the recognition of these graphs can be done in polynomial time for fixed k and is fixed parameter tractable when parameterized by k for graphs of bounded maximum degree.

6.1 Graphs G with $\nu(G) = \nu_2(G)$

In this section we review the results from [57, 19] and present shorter proofs.

A vertex of degree 1 in a graph G is a *leaf of* G. An edge of a graph G that is incident with a leaf of G is a *leaf edge*. An edge uv of a graph G such that uv belongs to a triangle in G, and u and v have degree 2 in G is a *triangle edge*.

The following lemma is a key observation from [57, 19]. For the sake of completeness, we include its short proof.

Lemma 6.1.1 (Kobler and Rotics [57], Cameron and Walker [19]). If G is a connected graph with $\nu(G) = \nu_2(G) \ge 2$, then every edge in a maximum induced matching is a leaf edge or a triangle edge.

Proof. Let M_2 be a maximum induced matching of G. Let e be an edge in M_2 . We may assume that e is not a leaf edge. If all other edges in M_2 are at distance at least 3 from e, then adding to M_2 an edge at distance 1 from e yields a matching of G with more edges

than M_2 , which is a contradiction. Hence, there is an edge f in M_2 at distance exactly 2 from e. Let $P: v_0v_1v_2v_3v_4$ be a path in G with $e = v_0v_1$ and $f = v_3v_4$. Since e is not a leaf edge and M_2 is an induced matching, the vertex v_0 has a neighbor u that is distinct from v_1, v_3 , and v_4 . If u is distinct from v_2 , then $(M_2 \setminus \{v_0v_1\}) \cup \{v_1v_2, v_0u\}$ is a matching of G with more edges than M_2 , which is a contradiction. Hence the only neighbors of v_0 are v_1 and v_2 . Considering the path $Q: v_1v_0v_2v_3v_4$ instead of P, it follows, by symmetry, that the only neighbors of v_1 are v_0 and v_2 , that is, the edge e is a triangle edge.

We proceed to Cameron and Walker's structural description of the graphs G with $\nu(G) = \nu_2(G)$.

If G is a connected graph of order at least 3, and V_1 , V_2 , V_3 , and V_4 are four disjoint and possibly empty sets that partition the vertex set of G, then (V_1, V_2, V_3, V_4) is a CWpartition of G if

- V_1 , V_2 , and V_3 are independent,
- V_4 induces a 1-regular subgraph $G[V_4]$ of G,
- V_1 is the set of leaves of G,
- V_2 is the set of neighbors of the vertices in V_1 ,
- every edge uv between vertices in V_4 is a triangle edge and the common neighbor of u and v lies in V_3 .

Note that all edges between V_1 and V_2 are leaf edges, every vertex of V_2 is incident with at least one leaf edge, and $V_2 \cup V_3$ induces a connected bipartite graph with partite sets V_2 and V_3 . If V_3 is empty, then V_2 contains exactly one vertex, V_4 is empty, and G is a star whose center is the unique vertex in V_2 . If V_1 is empty, then V_2 is empty, V_3 contains exactly one vertex, V_4 contains exactly two vertices, and G is a triangle. A set M of edges of a graph G with a CW-partition as above is a CW-matching of G if, for every triangle Cof G, the set M contains exactly one triangle edge of G that belongs to C, and, for every vertex v in V_2 , the set M contains exactly one leaf edge incident with v. For the complete graph K_2 of order 2, its unique edge is considered a CW-matching of K_2 . A graph G is a CW-graph if every component of G either has order at most 2 or has a CW-partition. It follows immediately that CW-graphs can be recognized efficiently.

A matching M of a graph G is *light* if no edge uv in M is adjacent to an edge vw of G such that \tilde{M} with $\tilde{M} = (M \setminus \{uv\}) \cup \{vw\}$ is a matching of G and the degree of w is smaller than the degree of u. Note that the degree sum over all vertices incident with the edges in \tilde{M} is smaller than that of M. Therefore, every matching can be transformed efficiently into a light matching with the same number of edges by a polynomial number of exchange operations.

The next result corresponds to the "if"-part of Theorem 1 in [19].

Lemma 6.1.2 (Cameron and Walker [19]). If G is a connected CW-graph, then $\nu_2(G) = \nu(G)$, and an induced matching of G is maximum if and only if it is a CW-matching.

Proof. Clearly, we may assume that G is neither a star nor a triangle. Let (V_1, V_2, V_3, V_4) be a \mathcal{CW} -partition of G. Let M be a light maximum matching of G. If M contains an edge e incident with a vertex in V_2 , then e is a leaf edge. If M contains an edge e that belongs to a triangle of G, then e is a triangle edge. Note that the sets $\{uv \in E(G) : u \in V_2\}$ of the edges of G incident with the vertices in V_2 together with the sets $\{uv, vw, wu\}$ of the edges of the triangles uvwu of G partition the edge set of G. Since every \mathcal{CW} -matching of G contains at least as many edges as M. Since every \mathcal{CW} -matching is an induced matching, it follows that $\nu_2(G) = \nu(G)$ and every \mathcal{CW} -matching of G is not a \mathcal{CW} -matching of G, then it contains an edge e incident with a vertex v in V_3 . Let \tilde{M} be a \mathcal{CW} -matching of G. Since I form v. Since M is induced, the set $(M \setminus \{e\}) \cup \{e_1, e_2\}$ is a matching of G with more edges than M, which is a contradiction. Hence every maximum induced matching of G is a \mathcal{CW} -matching of G. Since I form v. Since M is induced, the set $(M \setminus \{e\}) \cup \{e_1, e_2\}$ is a matching of G with more edges than M, which is a contradiction. Hence every maximum induced matching of G is a \mathcal{CW} -matching of G is a \mathcal{CW} -matching, which completes the proof.

The reason for the relevance of the notion of a light matching in this context is that a maximum matching of a connected CW-graph is a CW-matching if and only if it is light. By Lemma 6.1.2, this implies that a maximum induced matching of a CW-graph can be found efficiently.

The main result of Cameron and Walker [19] is the following.

Theorem 6.1.1 (Cameron and Walker [19]). If G is a graph, then $\nu_2(G) = \nu(G)$ if and only if G is a CW-graph.

Proof. Since the sufficiency follows from Lemma 6.1.2, we proceed to the proof of the necessity. Clearly, we may assume that G is a connected graph with $\nu(G) = \nu_2(G) \ge 2$. Let M_2 be a maximum induced matching of G. By Lemma 6.1.1, all edges in M_2 are leaf edges or triangle edges. Let V_1 be the set of leaves of G. Let V'_1 be the set of leaves of G that are incident with an edge in M_2 . Let V_2 be the set of neighbors of the vertices in V'_1 . Let V_4 be the set of vertices that are incident with a triangle edge in M_2 . Let $V_3 = V(G) \setminus (V_1 \cup V_2 \cup V_4)$. Since M_2 is an induced matching, the set V_2 is independent, and the common neighbor of adjacent vertices in V_4 lies in V_3 . Since G is connected and $\nu_2(G) \ge 2$, the set V_2 contains no leaf. If x is a leaf of G whose neighbor y is not in V_2 , then y is not in V_4 , and thus $M_2 \cup \{xy\}$ is a matching of G with more edges than M_2 , which is a contradiction. Hence, V_2 is the set of neighbors of the vertices in V_1 . If xy is an edge between two vertices in V_3 , then $M_2 \cup \{xy\}$ is a matching of G with more edges the proof.

6.2 Graphs G with $\nu(G) - \nu_2(G) \le k$

In this section we study graphs G with $\nu(G) - \nu_2(G) \leq k$. We collect some structural properties of these graphs in Section 6.2.1 and consider their recognition in Section 6.2.2.

6.2.1 Structure

Let G be a graph with $\nu(G) - \nu_2(G) \leq k$ for some non-negative integer k.

Let M_1 be a maximum matching of G and let M_2 be a maximum induced matching of G.

Let $H = (V(G), M_1 \Delta M_2).$

If some component of H is a path with 2 edges, say $e_1 \in M_1$ and $e_2 \in M_2$, then $(M_1 \setminus \{e_1\}) \cup \{e_2\}$ is a maximum matching of G having more edges in common with M_2 than M_1 . Iteratively applying this exchange operation to M_1 , we may assume that no component of H is a path with 2 edges. In this case we say that the maximum matching M_1 is close to the maximum induced matching M_2 . For the rest of the section, we assume that M_1 is close to M_2 .

Lemma 6.2.1. The components of H are

- isolated vertices,
- paths of length 1 whose edge belongs to $M_1 \setminus M_2$, and
- paths of length 3 whose two leaf edges belong to M₁ \ M₂ and whose middle edges belong to M₂ \ M₁.

Furthermore, H has exactly $\nu(G) - \nu_2(G)$ non-trivial components.

Proof. Since M_1 and M_2 are matchings, all components of H are M_1 - M_2 -alternating paths and cycles. Since M_2 is an induced matching, the graph H contains no path of length at least 4 and no cycle. Since M_1 is close to M_2 , and M_1 is a maximum matching, the components of H are as stated. Since every non-trivial component of H contains exactly one more edge from M_1 than from M_2 , it follows that H has exactly $\nu(G) - \nu_2(G)$ non-trivial components.

Let V_0 be the set of vertices of G that belong to a non-trivial component of H. Since every non-trivial component of H has order at most 4, Lemma 6.2.1 implies $|V_0| \le 4k$.

Let $V' = V(G) \setminus V_0$, $G_0 = G[V_0]$, and G' = G[V'].

Lemma 6.2.2. G' is a CW-graph.

Proof. Since no edge in $M_1 \cup M_2$ is between V_0 and V', we have $|M_i| = |M_i \cap E(G_0)| + |M_i \cap E(G')|$ for $i \in \{1, 2\}$. Note that $|M_1 \cap E(G_0)| = |M_2 \cap E(G_0)| + (\nu(G) - \nu_2(G))$,

 $|M_1 \cap E(G')| = |M_2 \cap E(G')|$, and that $M_2 \cap E(G')$ is an induced matching of G'. If G' is not a \mathcal{CW} -graph and N'_1 is a maximum matching of G', then, by Theorem 6.1.1, we have $|N'_1| = \nu(G') > \nu_2(G') \ge |M_2 \cap E(G')|$. Now $N_1 = (M_1 \cap E(G_0)) \cup N'_1$ is a matching of Gand hence

$$\begin{aligned}
\nu(G) &\geq |N_1| \\
&= |M_1 \cap E(G_0)| + |N_1'| \\
&= |M_2 \cap E(G_0)| + (\nu(G) - \nu_2(G)) + |N_1'| \\
&> |M_2 \cap E(G_0)| + (\nu(G) - \nu_2(G)) + |M_2 \cap E(G')| \\
&= |M_2| + (\nu(G) - \nu_2(G)) \\
&= \nu(G),
\end{aligned}$$

which is a contradiction.

6.2.2 Recognition

Theorem 6.2.1. For a fixed non-negative integer k, the graphs G with $\nu(G) - \nu_2(G) \le k$ can be recognized in polynomial time.

Proof. Let G be a graph of order n. Let M_1 be a maximum matching of G and M_2 be a maximum induced matching of G such that M_1 is close to M_2 . If $\nu(G) - \nu_2(G) \leq k$, then Lemma 6.2.1 implies $|M_1 \setminus M_2| \leq 2k$ and $|M_2 \setminus M_1| \leq k$, and we can consider all $n^{\mathcal{O}(k)}$ potential choices for $(M_1 \setminus M_2, M_2 \setminus M_1)$. Clearly, the components of $(V(G), M_1 \Delta M_2)$ must be as in Lemma 6.2.1. Furthermore, if $\nu(G) - \nu_2(G) \leq k$, then, by Lemmas 6.2.1 and 6.2.2, there must be a choice for $(M_1 \setminus M_2, M_2 \setminus M_1)$ such that the graph G' that arises from G by removing all vertices that are incident with an edge in $M_1 \Delta M_2$ has the following properties:

- (i) G' is a \mathcal{CW} -graph.
- (ii) There is a matching M' of G' such that
 - (a) M' is the union of \mathcal{CW} -matchings of the components of G'.
 - (b) $(M_2 \setminus M_1) \cup M'$ is an induced matching of G.
 - (c) $(M_1 \setminus M_2) \cup M'$ is a maximum matching of G.

As observed above, property (i) can be checked efficiently. Considering the edges between V(G') and the set of vertices of G that are incident with edges in $M_2 \setminus M_1$, it can be checked efficiently, if G' has a matching M' that satisfies (ii)(a) and (ii)(b). In fact, the necessary and sufficient conditions for the existence of such a matching are as follows: Every triangle of G' must contain a triangle edge that is not at distance 1 from any edge in $M_2 \setminus M_1$, and every vertex v that is incident with a leaf edge in G' must be incident with some leaf edge of G' that is not at distance 1 from any edge in $M_2 \setminus M_1$. Since all \mathcal{CW} -matchings of a connected \mathcal{CW} -graph have the same cardinality, the cardinality of M'is uniquely determined by G' and property (ii)(a). This implies that property (ii)(c) can be checked efficiently. If $\nu(G) - \nu_2(G) \leq k$, then one of the choices for $(M_1 \setminus M_2, M_2 \setminus M_1)$ must lead to an induced matching $(M_2 \setminus M_1) \cup M'$ of G with at least $\nu(G) - k$ edges, certifying that $\nu(G) - \nu_2(G) \leq k$. Conversely, if $\nu(G) - \nu_2(G) > k$, then this does not happen, which completes the proof.

Our next result states that the recognition of those graphs G with $\nu(G) - \nu_2(G) \leq k$ that are of bounded maximum degree is fixed parameter tractable when parameterized by k.

Theorem 6.2.2. Let Δ and k be positive integers. The graphs G with $\nu(G) - \nu_2(G) \leq k$ of maximum degree at most Δ can be recognized in $f(k, \Delta)n^c$ time where the constant c does not depend on Δ or k.

Proof. Let G be a graph of maximum degree at most Δ and let k be a positive integer. We consider an unknown yet fixed maximum induced matching M_2 of G. Our approach to decide whether $\nu(G) - \nu_2(G) \leq k$ relies on the construction of a rooted search tree T whose order is bounded in terms of k and Δ . With every vertex s of T, we associate a pair $(M_1(s), \mathcal{P}(s))$, where

- $M_1(s)$ is a maximum matching of G, and
- \$\mathcal{P}(s)\$ is a collection of paths in \$G\$ of lengths 1 and 3 whose leaf edges all belong to \$M_1(s)\$.

Let $V(\mathcal{P}(s))$ be the union of the vertex sets of the paths in $\mathcal{P}(s)$, let $M_1(\mathcal{P}(s))$ be the set of leaf edges of the paths in $\mathcal{P}(s)$, and let $M_2(\mathcal{P}(s))$ be the set of non-leaf edges of the paths in $\mathcal{P}(s)$. Note that $M_1(\mathcal{P}(s))$ is a perfect matching of the graph $G[V(\mathcal{P}(s))]$. Let $G'(s) = G[V(G) \setminus V(\mathcal{P}(s))]$ and $M'_1(s) = M_1(s) \setminus M_1(\mathcal{P}(s))$.

The collection $\mathcal{P}(s)$ encodes a potential choice for a collection of some non-trivial components of the graph H considered in Section 6.2.1, that is, in all situations represented by s and its descendants in T, we assume that

- $M_2(\mathcal{P}(s))$ is a (known) subset of the (unknown) matching M_2 ,
- $M_1(\mathcal{P}(s))$ is a subset of some maximum matching M_1 of G that is close to M_2 , and
- the paths in $\mathcal{P}(s)$ are components of $(V(G), M_1 \Delta M_2)$.

Note that M_1 is allowed to differ from $M_1(s)$ within G'(s).

Clearly, the set $M'_1(s)$ is a maximum matching of G'(s).

As an additional property of the pair $(M_1(s), \mathcal{P}(s))$ associated with s, we will ensure that $M'_1(s)$ is a light maximum matching of G'(s). As observed above, if $(M_1(s), \mathcal{P}(s))$ is such that $M'_1(s)$ is not a light maximum matching of G'(s), then $M'_1(s)$ can be transformed efficiently into a light maximum matching $\tilde{M}'_1(s)$ of G'(s). Replacing $M_1(s)$ with $\tilde{M}_1(s)$ where $\tilde{M}_1(s) = (M_1(s) \setminus M'_1(s)) \cup \tilde{M}'_1(s)$ ensures the desired lightness. For simplicity we say that $\tilde{M}_1(s)$ arises by *lightening* $M_1(s)$.

With the root r of T, we associate the pair $(M_1(r), \emptyset)$ where $M_1(r)$ is any light maximum matching of G. If t is a child of s in T, then $\mathcal{P}(t)$ contains exactly one more path than $\mathcal{P}(s)$. A vertex s of T is a leaf of T if and only if

- (i) either $\mathcal{P}(s)$ contains at most k paths, the graph G'(s) is a \mathcal{CW} -graph, and G'(s) has a maximum induced matching $M'_2(s)$ such that the set $M_2(s)$ with $M_2(s) = M_2(\mathcal{P}(s)) \cup M'_2(s)$ is an induced matching of G,
- (ii) or $\mathcal{P}(s)$ contains exactly k+1 paths. Let $M_2(s) = \emptyset$ in this case.

Note that no leaf of T has depth larger than k+1. The construction of T will ensure that

$$\nu(G) - \nu_2(G) \le k \quad \Leftrightarrow \quad \max\{|M_2(s)| : s \text{ is a leaf of } T\} \ge \nu(G) - k. \tag{6.1}$$

If $\nu(G) - \nu_2(G) \leq k$, then the largest induced matching of the form $M_2(s)$ where s is a leaf of T will certify $\nu(G) - \nu_2(G) \leq k$.

In order to complete the proof, we will describe the construction of T in such a way that

- (6.1) holds and
- for every vertex s of T,
 - $(M_1(s), \mathcal{P}(s))$ can be determined in time $\mathcal{O}(n^c)$ where c is a suitable constant,
 - the number of children of s is bounded in terms of Δ , and
 - if s is a leaf of T that satisfies (i), then $M'_2(s)$ can be found in time $\mathcal{O}(n^c)$.

Let s be a vertex of T that does not satisfy (i) or (ii), that is, s will not be a leaf of T. We consider two cases

Case 1 G'(s) is not a CW-graph.

In this case, since $M'_1(s)$ is a maximum matching of G'(s), Theorem 6.1.1 implies that $M'_1(s)$ is not induced. Therefore, there is a path $v_1v_2v_3v_4$ in G'(s) such that $v_1v_2, v_3v_4 \in M'_1(s)$.

It is possible that the edge v_2v_3 belongs to M_2 . In this case $v_1v_2v_3v_4$ is a path of length 3 in $(V(G), M_1(s)\Delta M_2)$. This possibility is reflected by creating a child t of s such that $\mathcal{P}(t) = \mathcal{P}(s) \cup \{v_1v_2v_3v_4\}$ and $M_1(t)$ arises by lightening $M_1(s)$. For the remaining children of s yet to be created in this case, we may assume that v_2v_3 does not belong to M_2 . Since M_2 is an induced matching, this implies that one of the two vertices v_2 and v_3 is not incident with an edge in M_2 . By symmetry, we may assume that v_2 is not incident with an edge in M_2 ; for the children of s yet to be created in this case, the tree T will contain symmetric children of s reflecting the possibility that v_3 is not incident with an edge in M_2 .

It is possible that no edge incident with v_1 belongs to M_2 . In this case v_1v_2 is a path of length 1 in $(V(G), M_1(s)\Delta M_2)$. This possibility is reflected by creating a child t of s such that such that $\mathcal{P}(t) = \mathcal{P}(s) \cup \{v_1v_2\}$ and $M_1(t)$ arises by lightening $M_1(s)$. For the remaining children yet to be created in this case, we may assume that some edge e^* of M_2 is incident with v_1 . Let N_1 be the set of neighbors v of v_1 in G'(s) that are incident with an edge vv' in $M'_1(s)$. Let N_2 be the set of neighbors of v_1 in G'(s) that are not in N_1 .

For every v in N_1 , it is possible that e^* is the edge v_1v . In this case v_2v_1vv' is a path of length 3 in $(V(G), M_1(s)\Delta M_2)$. This possibility is reflected by creating a child t of s such that $\mathcal{P}(t) = \mathcal{P}(s) \cup \{v_2v_1vv'\}$ and $M_1(t)$ arises by lightening $M_1(s)$. For the remaining children yet to be created in this case, we may assume that $e^* = v_1v$ for some v in N_2 .

For every v in N_2 , it is possible that e^* is the edge v_1v . Since $M'_1(s)$ is a light maximum matching of G'(s), the neighborhood of v in G'(s) is not a subset of $\{v_1, v_2\}$. This implies that v has a neighbor v' in $V(G'(s)) \setminus \{v_1, v_2\}$. If v' is not incident with an edge in $M'_1(s)$, then $M'_1(s) \cup \{vv'\}$ is a matching of G'(s) with more edges than $M'_1(s)$, which is a contradiction. Hence v' is incident with an edge v'v'' in $M'_1(s)$. Note that $(M_1(s) \setminus \{v'v''\}) \cup \{vv'\}$ is a maximum matching of G'(s). The possibility that e^* is the edge v_1v is reflected by creating a child t of s such that $\mathcal{P}(t) = \mathcal{P}(s) \cup \{v_2v_1vv'\}$ and $M_1(t)$ arises by lightening $(M_1(s) \setminus \{v'v''\}) \cup \{vv'\}$.

Case 2 G'(s) is a CW-graph.

Since $M'_1(s)$ is a light maximum matching of G'(s), it is the union of \mathcal{CW} -matchings of the components of G'(s). Since s does not satisfy (i) or (ii), the graph G'(s) does not have a maximum induced matching $M'_2(s)$ such that the set $M_2(s)$ with $M_2(s) =$ $M_2(\mathcal{P}(s)) \cup M'_2(s)$ is an induced matching of G. This implies that there is some component H'(s) of G'(s) such that every \mathcal{CW} -matching of H'(s) contains an edge at distance 1 from an edge in $M_2(\mathcal{P}(s))$. Since this property can be checked in $\mathcal{O}(n^c)$ time, such a component H'(s) can be found efficiently.

First we assume that H'(s) is a star or a triangle. In this case no edge of H'(s) can belong to M_2 but exactly one edge uv of H'(s) belongs to $M'_1(s)$. We create a child t of s such that $\mathcal{P}(t) = \mathcal{P}(s) \cup \{uv\}$ and $M_1(t)$ arises by lightening $M_1(s)$. Hence, we may assume that H'(s) is neither a star nor a triangle, which implies that it has order at least 4. Next we assume that some triangle edge uv of H'(s) is at distance 1 from an edge in $M_2(\mathcal{P}(s))$. Let w be the common neighbor of u and v in H'(s). It is possible that M_2 contains neither uw not vw. In this case uv is a path of length 1 in $(V(G), M_1(s)\Delta M_2)$. This possibility is reflected by creating a child t of s such that $\mathcal{P}(t) = \mathcal{P}(s) \cup \{uv\}$ and $M_1(t)$ arises by lightening $M_1(s)$. For the remaining children yet to be created in this subcase, we may assume, by symmetry, that vw belongs to M_2 . Since H'(s) is a \mathcal{CW} -graph of order at least 4 and $M'_1(s)$ is a light maximum matching of H'(s), there is a path uvwxy in H'(s) such that $xy \in M'_1(s)$ and $(M'_1(s) \setminus \{xy\}) \cup \{xw\}$ is a maximum matching of H'(s). The possibility that vw belongs to M_2 is reflected by creating a child t of s such that $\mathcal{P}(t) = \mathcal{P}(s) \cup \{uvwx\}$ and $M_1(t)$ arises by lightening $(M'_1(s) \setminus \{xy\}) \cup \{xw\}$. Hence, we may assume that no triangle edge of H'(s) is at distance 1 from an edge in $M_2(\mathcal{P}(s))$. Let V_1 be the set of leaves of H'(s) and let V_2 be the set of neighbors in H'(s) of the vertices in V_1 .

Next we assume that some vertex v in V_2 is at distance 1 from an edge in $M_2(\mathcal{P}(s))$. Clearly, $M'_1(s)$ contains an edge uv incident with v. Now uv is a path of length 1 in $(V(G), M_1(s)\Delta M_2)$. This possibility is reflected by creating a child t of s such that $\mathcal{P}(t) = \mathcal{P}(s) \cup \{uv\}$ and $M_1(t)$ arises by lightening $M_1(s)$. Hence, we may assume that no vertex in V_2 is at distance 1 from an edge in $M_2(\mathcal{P}(s))$. Since G'(s) does not have a maximum induced matching $M'_2(s)$ as in (i), this implies that there is some vertex v in V_2 such that all neighbors of v in V_1 are at distance 1 from an edge in $M_2(\mathcal{P}(s))$. Let uv be the edge of $M'_1(s)$ incident with v. It is possible that M_2 contains no edge incident with v. In this case uv is a path of length 1 in $(V(G), M_1(s)\Delta M_2)$. This possibility is reflected by creating a child t of s such that $\mathcal{P}(t) = \mathcal{P}(s) \cup \{uv\}$ and $M_1(t)$ arises by lightening $M_1(s)$. For the remaining children yet to be created in this subcase, we may assume that v is incident with an edge e^* in M_2 .

For every neighbor w of v in H'(s) such that w is not a leaf of H'(s), it is possible that e^* is the edge vw. Since H'(s) is a \mathcal{CW} -graph of order at least 4 and $M'_1(s)$ is a light maximum matching of H'(s), there is a path uvwxy in H'(s) such that $xy \in M'_1(s)$ and $(M'_1(s) \setminus \{xy\}) \cup \{xw\}$ is a maximum matching of H'(s). The possibility that e^* is the edge vw is reflected by creating a child t of s such that $\mathcal{P}(t) = \mathcal{P}(s) \cup \{uvwx\}$ and $M_1(t)$ arises by lightening $(M'_1(s) \setminus \{xy\}) \cup \{xw\}$.

This completes the construction of T. The number of children of every vertex of T is bounded in terms of Δ and on every child we spent $\mathcal{O}(n^c)$ time. Since the created children exhaust all possibilities, Lemmas 6.2.1 and 6.2.2 imply that (6.1) holds, which completes the proof.
6.3 Open Problem

In Theorem 6.2.2 we show that the graphs G with $\nu(G) - \nu_2(G) \leq k$ of maximum degree at most Δ can be recognized in $f(k, \Delta).n^c$ time where the constant c does not depend on Δ or k. Considering the maximum degree as unrestricted part of the input, we have the following open question.

• Can the graphs G with with $\nu(G) - \nu_2(G) \le k$ be recognized in $f(k).n^c$ time?

Chapter 7 Conclusions

In this thesis, a multivariate investigation of NP-hard problems has been carried out as a systematic application of classical and parameterized complexity techniques. This approach focused on drawing for each analyzed problem its boundaries between: (i) polynomial-time solvable and NP-hard subproblems; (ii) tractable and intractable parameterized versions. This strategy presents a more refined analysis of the complexity of problems, as well as their possibilities of solvability in practice. The idea is to map out when a problem Π becomes polynomial-time intractable and the sets of aspects that are responsible for this NP-hardness, i.e., the sets of aspects for which we can isolate its non-polynomial time complexity to solve Π as a purely function of them.

The tools used in this systematic analysis are first and foremost methods for algorithm design, or polynomial-time reductions (Karp reductions), to demonstrate polynomial time solvability or NP-hardness of a given problem, respectively. To show that a problem is fixed-parameter tractable, or in FPT, with respect to a set of parameters, we apply the bounded search tree and problem kernel (kernelization) methods. In another direction, we apply FPT-reductions (parametric reductions) to identify the level of parameterized intractability (W[t]-hardness, $t \geq 1$) of parameterized problems. Since every problem in FPT has a kernelization algorithm, we also analyze whether an FPT problem has a kernel of polynomial size with respect to its parameters. We establish the infeasibility of polynomial kernels for parameterized problems by a framework developed by Bodlaender et al. [11] and Fortnow and Santhanam [38], based upon the notion of or-compositionality, which shows that a problem does not admit a polynomial kernel unless $NP \subseteq coNP/poly$.

This thesis make a multivariate investigation of different groups of NP-hard problems: (i) and/or graph solution and its variants; (ii) flooding-filling games; (iii) problems on P_3 -convexity; (iv) problems on induced matchings.

1. And/or graph solution and its variants. We have proved that the problem MIN-AND/OR remains NP-hard even for and/or graphs where edges have weight one, or-vertices have out-degree at most two, and vertices with in-degree greater

than one are within distance at most one of a sink; and that deciding whether there is a solution subtree with weight exactly k of a given x-y tree is also NP-hard. In a parameterized point of view, we have shown that MIN-AND/OR⁰(k) is W[2]-hard, and MIN-X-Y(k) is W[1]-hard. We also deal with the main question: "Is MIN-AND/OR(k) \in FPT?". We answer positively to this question via a reduction to a problem kernel. Finally, we analyze whether MIN-AND/OR(k) admits a polynomial kernelization algorithm, and using the framework based upon the notion of orcompositionality, we show that MIN-AND/OR(k) does not admit a polynomial kernel unless $NP \subseteq coNP/poly$.

- 2. Flood-filling games. We analyze the complexity consequences of parameterizing Flood-it and Free-Flood-it by one or two of the following parameters: c - number of colors; λ - number of moves; d - maximum distance of the pivot (or diameter, in the case of Free-Flood-It); o - maximum orbit; k - number of leaves; r - number of bad moves. During our analysis we have shown that Flood-It on trees is analogous to Restricted Shortest Common Supersequence, and Flood-It remains NP-hard on 3-colored trees, closing an open question. We also present a general framework for reducibility from Flood-It to Free-Flood-It. Analyzing the computational complexity of these games on other classes of graphs such as powers of paths, power of cycles, circular grids, and graphs with bounded vertex cover, we conclude that: (i) Flood-it can be solved in polynomial time when played on P_n^2 , C_n^2 , and $2 \times n$ circular grids; (ii) Free-Flood-it is NP-hard when played on P_n^2 , C_n^2 ; and $2 \times n$ circular grids. Finally, we prove that Flood-it on graphs is fixed-parameter tractable considering the size of a minimum vertex cover as the parameter; in addition, we show a polynomial kernelization algorithm for Flood-it when, besides the minimum vertex cover, the number of colors is also a parameter.
- 3. Problems on P_3 -convexity. We prove that: (i) a minimum P_3 -hull set of a graph G can be found in polynomial time when $\delta(G) \geq \frac{n(G)}{c}$ (for some constant c); (ii) deciding if the size of a minimum P_3 -hull set of a graph is at most k remains NP-complete even on planar graphs with maximum degree four; (iii) a minimum P_3 -hull set of a cubic graph can be found in polynomial time; (iv) a minimum P_3 -hull set of bounded size and with no vertices of degree two; (v) deciding if the size of a minimum P_3 -geodetic set of a planar graph with maximum degree three is at most k is NP-complete. Some trivial parameterized results on P_3 -geodetic sets are also shown.
- 4. Problems on induced matchings. We present a short proof of a structural description of the graphs G where the matching number $\nu(G)$ equals the induced

matching number $\nu_2(G)$, and use it to study graphs G with $\nu(G) - \nu_2(G) \leq k$. We show that the recognition of these graphs can be done in polynomial time for fixed k, and is fixed parameter tractable when parameterized by k for graphs of bounded maximum degree. Finally, we extend some of Cameron and Walker's results to k-matchings in graphs of sufficiently large girth.

Bibliography

- ADELSON-VELSKY, G. M.; GELBUKH, A. F.; LEVNER, E. A Fast Scheduling Algorithm in AND-OR Graphs. In *Topics in Applied and Theoretical Mathematics* and Computer Science. WSEAS Press, 2001, 170–175.
- [2] ARTHUR, D.; CLIFFORD, R.; JALSENIUS, M.; MONTANARO, A.; SACH, B. The Complexity of Flood-Filling Games. 5th International Conference on Fun with Algorithms, FUN, Lecture Notes in Computer Science 6099 (2010), 307–318.
- [3] ASCHWANDEN, C. Spatial Simulation Model for Infectious Viral Disease with Focus on SARS and the Common Flu. 37th Annual Hawaii International Conference on System Sciences, HICSS (2004).
- [4] BALISTER, P. ; BOLLOBÁS, B. ; JOHNSON, J. ; WALTERS, M. Random Majority Percolation. *Random Structures and Algorithms* 36 (2010), 315–340.
- [5] BARBOSA, V. C. The Combinatorics of Resource Sharing. In Models for Parallel and Distributed Computation: Theory, Algorithmic Techniques and Applications. Kluwer Academic Publishers, 2002.
- [6] BARNETT, J. A.; VERMA, T. Intelligent Reliability Analysis. 10th IEEE Conference on Artificial Intelligence for Applications (1994), 428–433.
- [7] BARONE, P.; BONIZZONI, P.; VEDOVA, G. D.; MAURI, G. An Approximation Algorithm for the Shortest Common Supersequence Problem: An Experimental Analysis. *ACM Symposium on Applied Computing* (2001), 56–60.
- [8] BERMOND, J.-C.; BOND, J.; PELEG, D.; PERENNES, S. The Power of Small Coalitions in Graphs. *Discrete Applied Mathematics* 127 (2003), 399–414.
- [9] BOADLAENDER, H. L. Kernelization: New Upper and Lower Bound Techniques. 4th International Workshop on Parameterized and Exact Computation", IWPEC 2009, Lecture Notes in Computer Science 5987 (2009), 17–37.
- [10] BOADLAENDER, H. L.; BART, M. P. J.; KRATSCH, S. Kernel Bounds for Path and Cycle Problems. *Theoretical Computer Science* 511, 4 (2013).

- [11] BOADLAENDER, H. L.; DOWNEY, R. G.; FELLOWS, M. R.; HERMELIN, D. On Problems Without Polynomial Kernels. *Journal of Computer and System Sciences* 75 (2009), 423–434.
- [12] BOADLAENDER, H. L.; THOMASSÉ, S.; YEO, A. Kernel Bounds for Disjoint Cycles and Disjoint Paths. *Theoretical Computer Science* 412 (2011), 4570–4578.
- [13] BODLAENDER, H. L.; FELLOWS, M. R.; HALLETT, M. T.; WAREHAM, T.; WARNOW, T. The Hardness of Perfect Phylogeny, Feasible Register Assignment and other Problems on Thin Colored Graphs. *Theoretical Computer Science* 244 (2000), 167–188.
- [14] BRANDSTÄDT, A.; HOÁNG, C. Maximum Induced Matchings for Chordal Graphs in Linear Time. Algorithmica 52 (2008), 440–447.
- [15] BRANDSTÄDT, A.; MOSCA, R. On Distance-3 Matchings and Induced Matchings. Discrete Applied Mathematics 159 (2011), 509–520.
- [16] CAMERON, K. Induced Matchings. Discrete Applied Mathematics 24 (1989), 97–102.
- [17] CAMERON, K. Induced Matchings in Intersection Graphs. Discrete Applied Mathematics 278 (2004), 1–9.
- [18] CAMERON, K.; SRITHARAN, R.; TANG, Y. Finding a Maximum Induced Matching in Weakly Chordal Graphs. *Discrete Applied Mathematics* 266 (2003), 133–142.
- [19] CAMERON, K.; WALKER, T. The Graphs with Maximum Induced Matching and Maximum Matching the Same Size. Discrete Applied Mathematics 299 (2005), 49–55.
- [20] CAO, T.; SANDERSON, A. C. And/or Net Representation for Robotic Task Sequence Planning. IEEE Trans. Systems Man Cybernet, Part C: Applications and Reviews 28 (1998), 204–218.
- [21] CENTENO, C. C.; DOURADO, M. C.; PENSO, L. D.; RAUTENBACH, D.; SZWARC-FITER, J. L. Irreversible Conversion of Graphs. *Theoretical Computer Science* 412 (2011), 3693–3700.
- [22] CENTENO, C. C. ; PENSO, L. D. ; RAUTENBACH, D. ; DE SÁ, V. G. P. Immediate Versus Eventual Conversion: Comparing Geodetic and Hull Numbers in P₃-Convexity. 39th International Workshop on Graph-Theoretic Concepts in Computer Science, WG, Lecture Notes in Computer Science 7551 (2012), 262–273.
- [23] CENTENO, C. C.; PENSO, L. D.; RAUTENBACH, D.; DE SÁ, V. G. P. Geodetic Number versus Hull Number in P₃-Convexity. SIAM Journal on Discrete Mathematics 27, 2 (2013), 717–731.

- [24] CHANG, J.-M. Induced Matchings in Asteroidal Triple-Free Graphs. Discrete Applied Mathematics 132 (2003), 67–78.
- [25] CHOR, B.; FELLOWS, M. R.; RAGAN, M. A.; ROSAMOND, F. A.; SNIR, S. Connected Coloring Completion for General Graphs: Algorithms and complexity. 13th Conference on Computing and Combinatorics, COCOON, Lecture Notes in Computer Science 4598 (2007), 75–85.
- [26] CLIFFORD, R.; JALSENIUS, M.; MONTANARO, A.; SACH, B. The Complexity of Flood-Filling Games. *Theory of Computing Systems* 50, 1 (2012), 72–92.
- [27] CORANDI, R.; WESTFECHTEL, B. Version Models for Software Configuration Management. ACM Computing Surveys 30 (1998), 233–282.
- [28] DABROWSKI, K. K.; DEMANGE, M.; LOZIN, V. New Results on Maximum Induced Matchings in Bipartite Graphs and Beyond. *Theoretical Computer Science* 478 (2013), 33–40.
- [29] DELAVINA, E.; GODDARD, W.; HENNING, M. A.; PEPPER, R.; VAUGHAN, E. R. Bounds on the k-domination number of a graph. *Applied Mathematics Letters* 24, 6 (2011), 996–998.
- [30] DEMELLO, L. S. H.; SANDERSON, A. C. A Correct and Complete Algorithm for the Generation of Mechanical Assembly Sequences. *IEEE Trans. Robotics and Au*tomation 7 (1991), 228–240.
- [31] DOWNEY, R.; FELLOWS, M. Parameterized Complexity. Springer-Verlag, 1999.
- [32] DOWNEY, R. G.; FELLOWS, M. R. Parametrized Computational Feasibility. Progress in Computer Science and Applied Logic, Feasible Mathematics II 13.
- [33] DUCKWORTH, W.; MANLOVE, D.; ZITO, M. On the Approximability of the Maximum Induced Matching Problem. *Journal of Discrete Algorithms* 3 (2005), 79–91.
- [34] FELLOWS, M. R.; FERTIN, G.; HERMELIN, D.; VIALETTE, S. Sharp Tractability Borderlines for Finding Connected Motifs in Vertex-Colored Graphs. 34th International Colloquium on Automata, Languages and Programming, ICALP, Lecture Notes in Computer Science 4596.
- [35] FELLOWS, M. R.; HALLETT, M. T.; STEGE, U. Analogs and Duals of the MAST problem for Sequences and Trees. *Journal of Algorithms* 49, 1 (2003), 192–216.
- [36] FELLOWS, M. R.; HALLETT, M. T.; WAREHAM, H. T. DNA Physical Mapping: Three Ways Difficult. 1st Annual European Symposium on Algorithms, ESA, Lecture Notes in Computer Science 726 (1993), 157–168.

- [37] FLUM, J.; GROHE, M. Parameterized Complexity Theory. Springer, 2006.
- [38] FORTNOW, L. ; SANTHANAM, R. Infeasibility of Instance Compression and Succinct PCPs for NP. Journal of Computer and System Sciences 77 (2011), 91–106.
- [39] FREDKIN, E. Trie Memory. Communications of the ACM 3 (1960), 490–499.
- [40] GALLO, G.; LONGO, G.; NGUYEN, S.; PALLOTTINO, S. Directed Hypergraphs and Applications. Discrete Applied Mathematics 42 (1993), 177–201.
- [41] GAREY, M. R.; JOHNSON, D. S. Computer and intractability. A Guide to the NP-Completeness. Ney York, NY: WH Freeman and Company., 1979.
- [42] GAREY, M. R.; JOHNSON, D. S.; STOCKMEYER, L. Some Simplified NP-complete Problems. 6th Annual ACM Symposium on Theory of Computing (1974), 47–63.
- [43] GOLUMBIC, M.; KAPLAN, H.; SHAMIR, R. On the Complexity of DNA Physical Mapping. Advances in Applied Mathematics 15 (1994), 251–261.
- [44] GOLUMBIC, M.; LEWENSTEIN, M. New Results on Induced Matchings. Discrete Applied Mathematics 101 (2000), 157–165.
- [45] GOTTHILF, Z.; LEWENSTEIN, M. Tighter Approximations for Maximum Induced Matchings in Regular Graphs. 3rd Workshop on Approximation and Online Algorithms, WAOA, Lecture Notes in Computer Science 3879 (2006), 270–281.
- [46] GUO, J.; NIEDERMEIER, R. Invitation to Data Reduction and Problem Kernelization. ACM SIGACT News 38 (2007), 31–45.
- [47] GUSFIELD, D. Efficient Algorithms for Inferring Evolutionary Tree. Networks 21 (1981), 19–28.
- [48] HALLETT, M. T. An Integrated Complexity Analysis of Problems from Computational Biology, University of Victoria, Diss., 1996.
- [49] HANSBERG, A.; VOLKMANN, L. On graphs with equal domination and 2-domination numbers. *Discrete Mathematics* 308, 11 (2008), 2277–2281.
- [50] HANSBERG, A.; VOLKMANN, L. On 2-domination and independence domination numbers of graphs. Ars Combinatoria 101 (2011), 405–415.
- [51] HAYNES, T.; HEDETNIEMI, S.; SLATER, P. Fundamentals of Domination in Graphs. CRC Press, 1998.
- [52] J. BALOGH, B. B. Sharp Thresholds in Bootstrap Percolation. Physica A: Statistical Mechanics and its Applications 326 (2003), 305–312.

- [53] JIMÉNEZ, P.; TORRAS, C. Speeding up Interference Detection Between Polyhedra. *IEEE International Conference on Robotics and Automation* 2 (1996), 1485–1492.
- [54] JR, P. D.; ROBERTS, F. Irreversible k-Threshold Processes: Graph-Theoretical Threshold Models of the Spread of Disease and of Opinion. *Discrete Applied Mathematics* 157 (2009), 1615–1627.
- [55] KARP, R. M. Problem-Reduction Representations. In Problem Solving Methods in Artificial Intelligence. McGraw-Hill, 1971.
- [56] KARP, R. M. Reducibility Among Combinatorial Problems. In MILLER, R. E. (Hrsg.) ; THATCHER, J. W. (Hrsg.): Complexity of Computer Computations. Plenum Press, 1972.
- [57] KOBLER, D. ; ROTICS, U. Finding maximum induced matchings in subclasses of claw-free and P₅-free graphs, and in graphs with matching and induced matching of equal maximum size. Algorithmica 37 (2003), 327–346.
- [58] KUMAR, V. ; KANAL, L. N. Parallel Branch-and-Bound Formulations for And/Or Tree Search. Pattern Analysis and Machine Intelligence, PAMI, IEEE Transactions 6 (1984), 768–778.
- [59] LABER, E. S. A Randomized Competitive Algorithm for Evaluating Priced And/Or Trees. Theorical Computer Science 401, 1 (2008), 120–130.
- [60] LACROIX, V.; FERNANDES, C. G.; SAGOT, M. F. Motif Search in Graphs: Application to Metabolic Networks. *IEEE/ACM Transactions on Computational Biology* and Bioinformatics 3, 4 (2006), 360–368.
- [61] LICHTENSTEIN, D. Planar Satisfiability and its Uses. SIAM Journal on Computing 11 (1982), 329–343.
- [62] LOVÁSZ, L.; PLUMMER, M. Matching Theory. 29. Annals of Discrete Mathematics, North-Holland, Amsterdam, 1986.
- [63] LOZIN, V. On Maximum Induced Matchings in Bipartite Graphs. Information Processing Letters 81 (2002), 7–11.
- [64] MAIER, D. The Complexity of Some Problems on Subsequences and Supersequences. Journal of the ACM 25, 2 (1978), 322–336.
- [65] MCMORRIS, F. R.; WARNOW, T. J.; WIMER, T. Triangulating Vertex-Colored Graphs. SIAM Journal on Discrete Mathematics 7, 2 (1994), 296–306.

- [66] MEDEIROS, R. P.; SOUZA, U. S.; PROTTI, F.; MURTA, L. G. P. Optimal Variability Selection in Product Line Engineering. In Proc. of the 24th International Conference on Software Engineering and Knowledge Engineering - SEKE 2012, (2012), pp. 635– 640.
- [67] MEEKS, K.; SCOTT, A. The Complexity of Flood-Filling Games on Graphs. *Discrete* Applied Mathematics 160 (2012), 959–969.
- [68] MEEKS, K.; SCOTT, A. The Complexity of Free-Flood-It on 2 × n Boards. Theoretical Computer Science 500 (2013), 25–43.
- [69] MIDDENDORF, M. More on the Complexity of Common Superstring and Supersequence Problems. *Theoretical Computer Science* 125 (1994), 205–228.
- [70] MISRA, N. ; RAMAN, V. ; SAURABH, S. Lower Bounds on Kernelization. Discrete Optimization 8 (2011), 110–128.
- [71] MORABITO, R.; PUREZA, V. A Heuristic Approach Based on Dynamic Programming and And/Or-Graph Search for the Constrained Two-Dimensional Guillotine Cutting Problem. Annals of Operation Research 179 (2010), 297–315.
- [72] MORAN, S.; SNIR, S. Convex Recolorings of Strings and Trees: Definitions, Hardness Results and Algorithms. 9th International Workshop on Algorithms and Data Structures, WADS, Lecture Notes in Computer Science 3608 (2005), 218–232.
- [73] MOSER, H.; SIKDAR, S. The Parameterized Complexity of the Induced Matching Problem. Discrete Applied Mathematics 157 (2009), 715–727.
- [74] MOSER, H.; THILIKOS, D. Parameterized Complexity of Finding Regular Induced Subgraphs. Journal of Discrete Algorithms 7 (2009), 181–190.
- [75] NIEDERMEIER, R. Invitation to Fixed-Parameter Algorithms. Oxford Lecture Series in Mathematics and Its Applications, Oxford University Press, 2006.
- [76] PENSO, L. D.; PROTTI, F.; RAUTENBACH, D.; SOUZA, U. S. On P₃-convexity of Graphs with Bounded Degree. In 10th International Conference on Algorithmic Aspects of Information and Management, AAIM 2014, (2014), pp.
- [77] PIETRZAK, K. On the Parameterized Complexity of the Fixed Alphabet Shortest Common Supersequence and Longest Common Subsequence Problems. *Journal of Computer and System Sciences* 67 (4), 757–771.
- [78] R. FLEISCHER, G. J. W. An Algorithmic Analysis of the Honey-Bee Game. Theoretical Computer Science 452 (2012), 75–87.

- [79] RAHMANN, S. The Shortest Common Supersequence Problem in a Microarray Production Setting. *Bioinformatics* 19, Suppl. 2 (2003), ii156–ii161.
- [80] RAIHA, K.-J.; UKKONEN, E. The shortest Common Supersequence Problem Over Binary Alphabet is NP-complete. *Theoretical Computer Science* 16 (1981), 187–198.
- [81] SAHNI, S. Computationally Related Problems. SIAM Journal on Computing 3, 4 (1974), 262–279.
- [82] SIM, J.; PARK, K. The Consensus String Problem for a Metric is NP-complete. Journal of Discrete Algorithms 1, 1 (2003), 111–117.
- [83] SIMON, R.; LEE, R. C. T. On the Optimal Solution of And/Or Series Parallel Graphs. Journal of Association for Computing Machinery 18, 3 (1971), 354–372.
- [84] SOUZA, U. S. ; PROTTI, F. ; DANTAS DA SILVA, M. Complexidade Parametrizada para Problemas em Grafos E/OU. Pesquisa Operacional para o Desenvolvimento 4, 2 (2012).
- [85] SOUZA, U. S.; PROTTI, F.; DANTAS DA SILVA, M. Inundação em Grafos. In Proc. of the 16th Congreso Latino Iberoamericano de Investigación Operativa & 44th Simpósio Brasileiro de Pesquisa Operacional, CLAIO/SBPO 2012, (2012), pp.
- [86] SOUZA, U. S.; PROTTI, F.; DANTAS DA SILVA, M. Parameterized And/Or Graph Solution. In Proc. of the 12th Cologne Twente Workshop on Graphs and Combinatorial Optimization - CTW 2013, (2013), pp. 205–208.
- [87] SOUZA, U. S.; PROTTI, F.; DANTAS DA SILVA, M. Parameterized Complexity of Flood-Filling Games on Trees. 19th International Computing and Combinatorics Conference, COCOON, Lecture Notes in Computer Science 7936 (2013), 531–542.
- [88] SOUZA, U. S. ; PROTTI, F. ; DANTAS DA SILVA, M. Revisiting the Complexity of And/Or Graph Solution. Journal of Computer and System Sciences 79 (2013), 1156–1163.
- [89] STOCKMEYER, L. ; VAZIRANI, V. NP-Completeness of Some Generalizations of the Maximum Matching Problem. *Information Processing Letters* 15 (1982), 14–19.
- [90] UENO, S. ; KAJITANI, Y. ; GOTOH, S. On the Nonseparating Independent Set Problem and Feedback Set Problem for Graphs with no Vertex Degree Exceeding Three. *Discrete Mathematics* 38 (1988), 355–360.
- [91] WAREHAM, H. T. Systematic parameterized complexity analysis in computational phonology, University of Victoria, Diss., 1999.

[92] YAP, C.-K. Some Consequences of Non-Uniform Conditions on Uniform Classes. Theoretical Computer Science 26 (1983), 287–300.