Pursuit-Evasion, Decompositions and Convexity on Graphs

Ronan Pardo Soares

UNIVERSITÉ DE NICE - SOPHIA-ANTIPOLIS
ÉCOLE DOCTORALE DES SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET DE LA COMMUNICATION

UNIVERSIDADE FEDERAL DO CEARÁ
PROGRAMA DE MESTRADO E DOUTORADO EM CIÊNCIA DA COMPUTAÇÃO

P H D T H E S I S

to obtain the title of

Docteur en Sciences
de l'Université de Nice - Sophia Antipolis
Mention : Informatique

Doutor em Ciências
pela Universidade Federal do Ceará
Menção: Computação

Defended by
Ronan SOARES

Pursuit-Evasion, Decompositions and Convexity on Graphs

COATI Project
(Inria, I3S (CNRS/UNS))
Advisors:
David COUDERT
Nicolas NISSE

ParGO
Departamento de Computação
Advisor:
Cláudia LINHARES SALES

prepared at COATI Project (Inria, I3S (CNRS/UNS))
Defended on November 8, 2013

Jury:

Reviewers:	Dimitrios M. Thilikos	- Directeur de Recherche CNRS & U. of Athens
	Ioan Todinca	- Professor U. d'Orleans
	Jayme L. Szwarcfiter	- Professor U. Federal do Rio de Janeiro
Examinators:	David Coudert	- Chargé de Recherche Inria
	David Ilcinkas	- Chargé de Recherche CNRS
	Nicolas Nisse	- Chargé de Recherche Inria
	Cláudia Linhares Sales	- Professor U. Federal do Ceará
	Rudini Menezes Sampaio	- Professor U. Federal do Ceará

Acknowledgments

First, I am extremely indebted to all my advisers David Coudert, Nicolas Nisse and Cláudia Linhares Sales, for without their help, this work would not have been done. I am also grateful for all their teachings and remarks which allowed me to become a better researcher.

I dedicate this thesis to all my family, specially my parents, Alberto Melo Soares and Miriam Carmen Pardo Soares. They provided me with every kind of support necessary for my development and for that I am completely in their debt. I also dedicate this thesis to fiancee Tássia Gabrielle Ponte Carneiro. I could never thank her enough for all happiness that she brought into my life and for all the support and understanding she dedicated to me.

I also give a special thanks to Patricia Lachaume who went beyond the call of duty and aided me immensely with all my administrative problems and even personal ones. I would like to give another special thanks to my dear friends Julio César Silva Araújo, Ana Karolinna Maia de Oliveira and Leonardo Sampaio Rocha. Their assistance and companionship greatly helped me feeling at home.

I could not mention everyone, but I also thank everyone from MASCOTTE/COATI for providing a fantastic workplace. I hope they can always keep their joyous nature.

Sophia-Antipolis, France
October 17, 2013

Pursuit-Evasion Games, Graph Decompositions and Convexity in Graphs

Abstract:

This thesis focuses on the study of structural properties of graphs whose understanding enables the design of efficient algorithms for solving optimization problems. We are particularly interested in methods of decomposition, pursuit-evasion games and the notion of convexity.

The Process game has been defined as a model for the routing reconfiguration problem in WDM networks. Often, such games where a team of searchers have to clear an undirected graph are closely related to graph decompositions. In digraphs, we show that the Process game is monotone and we define a new equivalent digraph decomposition. Then, we further investigate graph decompositions. We propose a unified FPT-algorithm to compute several graph width parameters. This algorithm turns to be the first FPT-algorithm for the special and the q-branched tree-width of a graph.

We then study another pursuit-evasion game which models prefetching problems. We introduce the more realistic online variant of the Surveillance game. We investigate the gap between the classical Surveillance Game and its connected and online versions by providing new bounds. We then define a general framework for studying pursuit-evasion games, based on linear programming techniques. This method allows us to give first approximation results for some of these games.

Finally, we study another parameter related to graph convexity and to the spreading of infection in networks, namely the hull number. We provide several complexity results depending on the graph structures making use of graph decompositions. Some of these results answer open questions of the literature.

Keywords: Graph Searching, Pursuit-Evasion Games, Graph Decompositions, Cops and Robber, Surveillance Game, Convexity, Hull Number.

Jeux de Poursuite-Evasion, Décompositions et Convexité dans les Graphes

Résumé :

Cette thèse porte sur l'étude des propriétés structurelles de graphes dont la compréhension permet de concevoir des algorithmes efficaces pour résoudre des problèmes d'optimisation. Nous nous intéressons plus particulièrement aux méthodes de décomposition des graphes, aux jeux de poursuites et à la notion de convexité.

Le jeu de Processus a été défini comme un modèle de la reconfiguration de routage. Souvent, ces jeux où une équipe de chercheurs doit traiter les sommets d'un graphe non orienté sont reliés aux décompositions de graphes. Dans les graphes dirigés, nous montrons que le jeu de Processus est monotone et nous définissons une nouvelle décomposition de graphes que lui est équivalente. Ensuite, nous étudions d'autres décompositions de graphes. Nous proposons un algorithme FPT unifié pour calculer plusieurs paramètres de largeur de graphes. En particulier, ceci est le premier FPT-algorithme pour la largeur arborescente q-branchée et pour la largeur arborescente spéciale d'un graphe.

Nous étudions ensuite un autre jeu qui modélise les problèmes de pré-chargement. Nous introduisons la variante en ligne du jeu de surveillance. Nous étudions l'écart entre le jeu de surveillance classique et ses versions connectée et en ligne, en fournissant de nouvelles bornes. Nous définissons ensuite un cadre général pour l'étude des jeux poursuite-évasion. Cette méthode nous permet de donner les premiers résultats d'approximation pour certains de ces jeux.

Finalement, nous étudions un autre paramètre lié à la convexité des graphes et à la propagation d'infection dans les réseaux, le enveloppe convexe. Nous fournissons plusieurs résultats de complexité en fonction des structures des graphes et en utilisant des décompositions de graphes.

Mots clés : Gendarmes et Voleurs, Jeux de Évasion et Poursuite, Decomposition de Graphes, Jeu de Surveillance, Convexité, Enveloppe Convexe.

Jogos de Perseguição-Evasão, Decomposições e Convexidade em Grafos

Resumo:

Esta tese é centrada no estudo de propriedades estruturais de grafos cujas compreensões permitem a concepção de algoritmos eficientes para resolver problemas de otimização. Estamos particularmente interessados em decomposições, em jogos de perseguição-evasão e em convexidade.

O jogo de Processo foi definido como um modelo para a reconfiguração de roteamento em redes WDM. Muitas vezes, jogos de perseguição-evasão, em que uma equipe de agentes tem como objetivo limpar um grafo não direcionado, estão intimamente relacionados com decomposições em grafos. No caso de grafos direcionados, mostramos que o jogo de Processo é monotônico e definimos uma nova decomposição em grafos equivalente a tal jogo. A partir de então, investigamos outras decomposições em grafos. Propomos um algoritmo FPT para calcular vários parâmetros de largura em grafos. Em particular, este é o primeiro algoritmo FPT para calcular a largura em árvore especial e a largura em árvore q-ramificada de um grafo.

Em seguida, estudamos um outro jogo perseguição-evasão que modela problemas de pré-obtenção. Nós introduzimos uma versão mais realista do jogo de Vigilância a versão on-line. Estudamos a diferença entre o jogo de Vigilância clássico e suas versões conectadas e on-line, fornecendo novos limites para essa diferença. Nós, então, definimos um modelo geral para o estudo de jogos perseguição-evasão, com base em técnicas de programação linear. Este método permite-nos dar os primeiros resultados de aproximação para alguns desses jogos.

Finalmente, estudamos outro parâmetro relacionado com a convexidade e a propagação da infecção em redes, o "hull number". Nós fornecemos vários resultados de complexidade computacional, dependendo das propriedades estruturais do grafo de entrada e usando decomposições em grafos. Alguns destes resultados respondem problemas em aberto na literatura.

Palavras-chave: Procura em Grafos, Jogos de Perseguição-evasão, Decomposição em Grafos, Jogo de Observação, Convexidade, "Hull Number".

Contents

Chapter 1

Introduction

This thesis is dedicated to the study of structural properties of graphs and how they can be used to aid the design of efficient algorithms for problems arising in telecommunication networks. By structural properties, we mainly refer to properties that do not depend on the representation of the graph. Unfortunately, there is no common formal definition of a structural property of a graph. We can, however, illustrate this notion with some examples: one simple example of structural property is the exclusion of cycles. That is, the family of graphs known as forests. Another important structural property is planarity, which gave rise to the notion of graphs with bounded genus. In other words, graph structural properties are the ones that can be described by either excluding a pattern, for example forbidding the graph to have some other graph as a minor or as a subgraph, or by enforcing the graph to have some underlying structure, for example be embeddable into a given surface, have some bounded parameter, or being connected. Note that the properties of being embeddable into a given surface and excluding some other graphs as minors are equivalent.

Since there are little hopes in finding polynomial time algorithms for hard problems such as NP-complete or PSPACE-complete problems, one possible alternative to tackle them is to study their hardness when the inputs are restricted to a certain class. That is, in the case of problems in graph theory, sometimes it is possible to solve, efficiently, hard problems for some graph classes. For example, while vertex coloring is NP-complete when the input graph is arbitrary, it becomes trivial in the class of bipartite graphs. In other words, knowing that the input graph does not have odd cycles, the problem of computing a proper coloring with minimum number of colors becomes trivial. Another example can be found in the traveling salesman problem. That is, while this problem is APX-complete[1] for general graphs, it has a polynomial time approximation scheme when the graph is euclidean and planar.

We investigate problems arising in telecommunication networks by focusing on the study of three wide and fundamental subjects in graph theory: graph decompositions, pursuit-evasion games and graph convexity. Our general approach, mainly, is to examine how graph structures may help understanding or solving problems in these subjects.

[1] Unless P = NP, there are no polynomial time approximation schemes for problems which are APX-complete.

1.1 Graph Decompositions

Graph decompositions are closely related to graph structural properties. Informally, a graph decomposition can be described as a family of subsets of vertices, or edges, of the graph that are organized in a particular manner. For example, in the tree decomposition [RS84], each subset in this family is a separator and the family is organized in a tree-like manner. This decomposition allows us to prove properties or to run algorithms for general graphs using techniques somewhat similar to the ones used for trees. We give more details later in this section.

Another famous example of graph decomposition, the modular decomposition [Gal67], is based on the fact that maximal modules[2] of a graph can be organized into an hierarchical structure.

In general, the main purpose of graph decompositions is to split the graph into smaller pieces while some properties, depending on the decomposition used, are preserved. This can be of great help in order to apply techniques based on a divide and conquer paradigm.

One example of such technique is the dynamic programming method. This is a method for solving problems, by recursively breaking the problem one wants to solve into smaller sub-problems, solving each sub-problem and, then, combining up the solutions of these sub-problems. The key ingredients in dynamic programming are the existence of an efficient algorithm to combine the solutions for each of the sub-problems and a limit on the amount of different sub-problems.

To illustrate such method we show how to use dynamic programming to compute the maximum independent set[3] of a tree in polynomial time.

Let $T = (V, E)$ be a tree and $r \in V$ be any vertex of T. We root T in r. A maximum independent set of T either contains r, in which case it does not contain any children of r, or it does not contain r, in which case it can be obtained from the union of maximum independent sets for each subtree of T rooted on a child of r. The main idea is that we can combine the solutions for the maximum independent sets of subtrees rooted in each child of a given node $i \in V$ to obtain a maximum independent set for the subtree rooted at i. The reason that these solutions can be combined is mainly due to the fact that they overlap only on a single vertex, the root of the subtree. Then, by combining the solutions for the children of r we have the maximum independent set of T.

There are plenty of other problems that can be approached by using dynamic programming "guided" by graph decompositions. For example, algorithms to solve the maximum independent set problem, the maximum dominating set problem and minimum vertex coloring problem were given in [AP89]. Without being extensive, there are also algorithms to solve the tour merging problem [CS03], the call routing problem [ST94] and the disjoint paths problem [RS95]. In fact, every graph property definable in monadic second-order logic can be decided in linear time on graphs of bounded tree-width [Cou89]. Albeit the algorithms are, mostly, polynomial in the size of the input graph, they are exponential in a parameter of the decomposition of the graph used by the algorithm, namely, its width, which is, roughly speaking, the cardinality of the greatest subset of the family defining the decomposition. In other words, these algorithms are polynomial for classes of graphs which have a bounded width. Often, these algorithms are fixed parameter tractable (FPT)[4] where the parameter is the width of the graph. For example, outer-planar graphs have tree width two.

[2] A set of vertices that have the same neighborhood outside this set.

[3] An independent set $S \subseteq V(G)$ of a graph G is such that, for any $\{x, y\} \subseteq S$, $\{x, y\} \notin E(G)$.

[4] An algorithm is FPT with parameter k, if its time complexity can be bounded by $f(k) \cdot n^{O(1)}$, where n is

Since many of the algorithms that use graph decompositions have either to be given the decomposition as input or to compute it, one of the main challenges on graph decompositions is computing a decomposition of a graph. Moreover, since the complexity of these algorithms, sometimes, depends on the width of the decomposition, it is important to compute a graph decomposition minimizing its width. Unfortunately, there are several graph decompositions in which finding such good decomposition is NP-hard. For example, finding a good tree decomposition [AP89], a good path decomposition [ACP87], a good branch decomposition [ST94], or a good linear decomposition [Thi00] of a general graph is NP-hard. On the other hand, a modular decomposition of any graph can be found in linear time [TCH08].

Graph decompositions are not only important due to their relationship with dynamic programming methods to solve problems, but also as tools to prove important structural results in graph theory. As an example, we can cite the crucial role of tree decompositions in the proof of Wagner's conjecture [RS04] as well as the role of the modular decomposition in the proof of the strong perfect graph theorem [CRST06]. Wagner's conjecture stated that the undirected graphs, partially ordered by the graph minor relationship, form a well quasi-ordering. Planarity, for example, is a structural property and can be characterized by a set of forbidden minors, which is also a consequence of the Wagner's conjecture. It is well known that graphs that are planar are exactly the ones that do not have neither $K_{3,3}$, the complete bipartite graph with three vertices in each partition, nor K_5, the complete graph with five vertices, as minors [Kur30].

Due to their importance, one of our goals is to investigate the complexity of computing graph decompositions. In the literature, there are FPT-algorithms for computing tree, path, branch, linear, carving or cut decompositions of a graph where the parameter is the width of the decomposition. Most of these algorithms are based on the dynamic programming method, but with some technical differences. More precisely, we aim at investigating if it is possible to design an unified FPT-algorithm for computing all the aforementioned decompositions, while also including other graph decompositions that could benefit from such approach.

1.2 Pursuit-evasion Games

The second axis of our research, the pursuit-evasion games, is a collection of games played by two players, such that one takes the role of a pursuer while the other takes the role of an evader. Pursuit-evasion games, often, have a close relationship with graph decompositions, which will be explained later in this section. In all these games, each player plays by moving, adding or removing tokens that are on vertices or edges of the graph. While most of the games share a similar set of rules on how these tokens are added, moved or removed and on how each of the players win the game, they can differ significantly.

A classical pursuit-evasion game is the one known as the *Helicopter Search*, or *Node Search*, defined in [ST93]. In this game, cops and a robber occupy vertices of the graph. Both players, the cops and the robber, always know the other player's position. While the cops move by jumping from one vertex to the other in the graph, the robber moves by following paths free of cops. The objective of the cops is to capture the robber, by moving to its current position while, at the same time, blocking its escape. The robber wins the game if it is able to avoid capture indefinitely. It is clear that n cops are sufficient to

the size of the input and $f(k)$ is an arbitrary function depending only on k.

capture a robber on any graph of order n. If n cops are available, they win by occupying each vertex of the graph. For some graphs, this amount of cops can be rather excessive. For example, on any tree two cops are enough to guarantee the capture of the robber.

Hence, one main interest in these games is, usually, to determine the minimum amount of resources necessary to guarantee the victory for the pursuer, since it usually trivial for a pursuer with unlimited resources to win against the evader. For example, in the case of the the Helicopter Search game, while n cops are able to capture any robber in any graph of order n, two cops cannot guarantee the capture of a clever robber in a cycle. Unfortunately, computing the minimum amount of resources necessary to ensure the victory of the pursuer is a NP-hard problem for several pursuit-evasion games [KP86, MHG$^+$81, Thi00, GR95, FGP12] or even PSPACE-hard [Mam13].

These games have a wide variety of applications. Without being exhaustive, we mention some of them. One first example is the search for a deranged explorer in a maze of caves [Par78]. This problem can be modeled as a pursuit-evasion game by considering the maze of caves as a graph, the deranged explorer as the evader and the searchers as the pursuers. This reasoning can also be used to model the problem of eliminating a virus from a computer network [Als04], by considering the virus as the evader and "anti-virus programs" as the pursuers. A graph decomposition that can help to solve the problem of compact routing was designed with the help of a pursuit-evasion game [KLNS12]. Finally, we can also cite the Process game, which models the problem of routing reconfiguration in WDM networks [CPPS05], and the Surveillance game, which models prefetching problems [FGJM$^+$12]. In particular, the Surveillance game was inspired by the problem of prefetching web-pages from the world wide web, in order to minimize the time a web-surfer waits for a web-page to be downloaded.

In addition to their applications, some pursuit-evasion games also have a close relationship with graph decompositions. For example, the minimum number of cops such that the cops can always guarantee the capture of a visible robber in the Helicopter Search game is equal to the tree-width of the graph plus one. This is mainly due to the monotonicity of some pursuit-evasion games. A pursuer playing monotonously is forbidden to put a token on a vertex of the graph after it has removed all its tokens from this vertex on a previous move. For example, in the Helicopter Search game, this means that once a cop leaves a vertex, no other cop can occupy this vertex for the remainder of the game. A pursuit-evasion game is said to be monotone if a pursuer playing in a monotone way does not need more resources to win against the evader than a pursuer without this constraint. For example, in the Helicopter Search game, k cops can capture the robber on a graph if, and only if, k cops playing monotonously can capture the robber on the same graph. Games that are monotone often have a close relationship with some graph decompositions [ST93, SB91, KC85]. This is due to (1) the equivalence between monotone strategies for the pursuer and the particular decomposition associated with the game and (2) the fact that restricting the game to be played only with monotone strategies does not increase the amount of resources necessary for the pursuer to win. In other words, monotone strategies for the pursuer are simply a different manner to represent a particular graph decomposition.

If, on the one hand, monotone games have often a close relationship with some particular graph decomposition, on the other hand, not all games are monotone. In particular, some pursuit-evasion games in directed graphs are not monotone [Adl07, KO08], while other are [Bar06, YC07].

In either cases, monotone strategies in pursuit-evasion games in directed graphs are

often equivalent to some particular directed graph decomposition. One of the main challenges is to design graph decompositions for directed graphs that are as powerful as graph decompositions are to undirected graphs. Graph decompositions, to be considered "good", should have two main properties: (1) be algorithmically useful and (2) have nice structural properties, such as being closed under taking subdigraphs and some form of arc contractions [GHK+10]. Unfortunately, non-monotone pursuit-evasion games are often associated with graph decompositions that are not closed under taking subdigraphs and some form of arc contractions.

Moreover, monotonicity provides a simple certificate to show that a pursuit-evasion problem is in NP, since a graph decomposition associated with the pursuit-evasion game can be the certificate, if it has polynomial size.

We aim at investigating the property of monotonicity in the Process game and its relationship with directed graph decompositions. We are also interested in other pursuit-evasion games related to problems in telecommunication networks such as the Surveillance Game. In particular, we aim at investigating the relationship between the Surveillance game and its connected version.

1.3 Convexity

In Euclidean spaces, a set S of points is convex if, for every $a, b \in S$, we have that all the points that lie on a straight line between a and b also belong to S. For example, all simple regular polygons are convex. Convexity allows us to describe infinite sets efficiently, or in a compact manner. For example, take any convex set S and let S' be the set of its vertices, then any point x in S can be described as a convex combination of points in S' and every convex combination of points in S' is a point in S. Hence, convex infinite sets can be represented by its set of vertices which, hopefully, is not infinite. The last axis of research studied in this thesis is the concept of convexity applied to graphs.

One classical example of the concept of convexity when applied to graphs is the *geodetic convexity*, or the shortest path convexity. A subset S of the vertices of a graph is convex if all vertices in a shortest path between two elements of S also belong to S. A hull set S of a graph $G = (V, E)$ is a subset S of V such that the minimum convex set S' that contains S is V, that is, $S' = V$. The (geodetic) hull number of a graph $G = (V, E)$ is the minimum cardinality of a hull set of G.

The process to obtain a hull set of a graph can be seen as an iterative process as follows. Start with any subset S of vertices of the graph. Until no more vertices can be added to S, add to S the vertices of G that lie in any shortest path between any two vertices of S. S is a hull set of G if, and only if, at the end of this process, S is the vertex set of G.

One application related to graph convexities is the inference scheme in the normalization process of databases [KN13]. If all functional dependencies, in a relational database, are of the type $AB \to C$ and these functional dependencies can be modeled as a graph where the vertex C is in the middle of a shortest path joining A to B, then every hull set of this graph is a candidate key[5] for this database.

Another application related to graph convexities, namely the P_3-convexity, is the spread of infection on a network [CDD+10]. During the iterative process of obtaining a hull set in the P_3-convexity, instead of adding vertices to S based on any shortest path,

[5] A candidate key in a database is a set of attributes that can uniquely define a tuple of a table. They play an important role in the normalization process of a database.

we only consider paths of length 2. If we assume that the network is represented by a graph, then, in the spread of infection on a network, a node in the network becomes infected if at least two of its neighbors are infected. Therefore, the problem of knowing the minimum number of nodes of a network that are necessary and sufficient to infect all of its nodes is equivalent to computing the (P_3)-hull number of the graph representing this network.

While the problem of computing the (geodetic) hull number of a graph is NP-hard [DGK+09], for some particular well structured graphs, this problem can easily be solved. For example, there are algorithms to compute the hull number of a graph in polynomial time if the graph is either a cograph, a split graph, an unity interval graph [DGK+09], a distance hereditary graph or a chordal graph [KN13].

In this thesis, we aim at investigating how some structural properties of a graph affect the hardness of computing its hull number and how these properties can be used to obtain bounds for it. In particular, we investigate how some graph decompositions, such as the modular decomposition, can be used to aid in the computation of the hull number of some graphs.

1.4 Main Contributions and Outline

This thesis is divided in three parts. The first part, formed by Chapters 2, 3 and 4, studies some pursuit-evasion games, known as graph searching games, and graph decompositions. The second part of this thesis, formed by Chapters 5, 6 and 7, is dedicated to the study of turn-by-turn pursuit-evasion games. Lastly, the third part, formed by Chapters 8 and 9, studies the concept of convexity on graphs.

In **Chapter 2**, we deepen the study of some pursuit-evasion games, known as graph searching games, and graph decompositions, by formalizing these concepts and stating some of the most important results in this area.

Chapter 3 is dedicated to the study of a pursuit-evasion game known as the *Process* game. We first investigate the role of monotonicity in the Process game. We aim at answering how useful backtracking is in the problem of routing reconfiguration in WDM networks. We show that allowing recontamination does not help the searchers. In other words, we show that the Process game is monotone.

We also design a decomposition of the graph, the Process Decomposition, that is equivalent to monotone strategies for the Process game. Meaning that the problem of routing reconfiguration can be restated as the problem of computing a Process Decomposition. The results in Chapter 3 can be found in [NS13].

In **Chapter 4**, we proceed with the investigation of the problem of computing graph decompositions. We propose an unified FPT-algorithm to compute several graph decompositions (such as, tree decomposition, path decomposition, branch decomposition, linear decomposition, cut decomposition and carving decomposition). Moreover, this algorithm is the first to compute the special tree decomposition and any q-branched version of the aforementioned decompositions. This algorithm is based on the representation of these decompositions with partitioning functions and a dynamic programming approach based on an efficient representation of these partitioning functions.

The second part of this thesis starts with **Chapter 5**. In this chapter, we properly define turn-by-turn pursuit evasion games such as the cops and robbers game, the Angel Problem, the Eternal Dominating Set and the Eternal Vertex Cover, while also giving a brief survey about these games.

Then, in **Chapter 6**, we further investigate another pursuit-evasion game, related to prefetching problems, the Surveillance Game. This is a turn-by-turn game, where the pursuer plays by marking vertices of the graph and the evader plays by moving along at most one edge during its turn. The objective of the evader is to reach any vertex devoid of marks, while the pursuer wants to mark every vertex of the graph before the evader reaches any unmarked vertex. We define the Online variant of the Surveillance game, which models the problems of prefetching more realistically by imposing that the pursuer discovers the graph as the game progresses.

We continue by investigating the relationships between the "classical" Surveillance game, the Connected Surveillance game and the Online Surveillance game. More precisely, we aim at answering how big can be the gap between the number of marks per turn necessary to guarantee a victory for the observer between these games. We show that, unfortunately, the best online strategy is to mark neighbors of the evader's position at each step. For the connected variant, we improve known upper and lower bounds for this gap. Results in Chapter 6 can be found in [GMN$^+$13].

Then, in **Chapter 7**, we study general turn-by-turn pursuit-evasion games. We propose a framework to relax the constraint that tokens used by both players must be integral. In other words, we allow both players to move and use parts of a token. Pursuit-evasion games that can be described using this framework includes, but is not limited to, several variants of the cops and robbers game, the Angel Problem game, the Eternal Dominating Set game and the Surveillance game. We aim at analyzing the behavior of such games when the integrality of the tokens for each player, or just for the pursuer, is relaxed.

We provide an algorithm to decide whether the pursuer has a winning strategy against the evader for any game that fit in this framework. This is achieved by considering the game as a convex game and applying linear programming techniques. These fractional games are also shown to give lower bounds to their integral versions. These lower bounds also allows us to develop the first approximability results for the Surveillance Game and the Angel Problem. Some results in Chapter 7 were presented in Algotel2013 [Soa13].

In the first chapter of the third part, **Chapter 8**, we give a brief survey on the concept of convexity when applied to graphs.

Then, in **Chapter 9** we study the computational complexity of computing the hull number of a graph in an attempt to pinpoint where does the hardness of computing this parameter lies. We start by answering an open question in [DGK$^+$09] by showing that computing the hull number of a bipartite graph is NP-hard. We proceed to consider this question in other graph classes such as complement of bipartite graphs, $(q, q-4)$-graphs and $\{P_5, K_3\}$-free graphs. We propose polynomial algorithms to compute the hull number of any graph belonging to these classes. These algorithms are, usually, based on graph decompositions.

We also propose the first FPT-algorithm for computing the hull number of general graphs, where the parameter is either their minimum vertex cover or their neighborhood diversity. Moreover, the techniques used to design this FPT-algorithm also allows us to characterize the hull number of the lexicographic products of graphs based on the hull number of its factors. The results in Chapter 9 are a compilation of the results found in [ACG$^+$11], [AMS$^+$13] and [ACG$^+$13].

Finally, in **Chapter 10**, we review the most important results in this thesis, while proposing directions for future work in these areas.

1.5 Basic Terminology

In this section, common definitions and notations of graph theory, necessary to properly understand this thesis, are recalled. For more definitions, we refer the reader to [BM08].

An undirected *graph* $G = (V, E)$ is defined by a non-empty set, V (or $V(G)$) of elements called *vertices*, a set E (or $E(G)$) of *edges* and a function $\rho_G : E \to (V \times V)$ that attributes a non-ordered pair of vertices of G to each edge in E. To simplify the notation, if $e \in E$ and $\rho_G(e) = (u, v)$, then we write $e = (u, v)$ or $e = uv$, in the case that there is no ambiguity. Two vertices u and v are *adjacent* or *neighbors*, if there exists an edge uv in E. The *extremities* of an edge $uv \in E$ are u and v. If u and v are extremities of e, then e is *incident* to u and v. If $uv \in E$ and $u = v$, we say that e is a *loop*. If there are more than two edges with the same extremities, we say that they are *multiple edges*. A graph $G = (V, E)$ without loops or multiple edges and with finite V is called a *simple graph*. In this case, the function ρ_G can be omitted from the description of the graph.

If the set of edges is formed by ordered pairs, we say that the graph is an *oriented graph*, a *directed graph* or, simply, a *digraph*.

The *degree*, $d(v)$, of a vertex v is the number of edges that are incident to v, each loop being counted twice. The smallest (resp. biggest) degree of a vertex in G is denoted by $\delta(G)$ (resp. $\Delta(G)$). The *neighborhood* of a vertex v in $G = (V, E)$ is the set $N(v) = \{u \mid (u, v) \in E\}$. If S is a subset of V, then $N(S) = \bigcup_{v \in S} N(v)$.

In a directed graph $G = (V, E)$, the *out-neighborhood*, $N^+(v)$, of a vertex v is given by $\{u \mid (v, u) \in E\}$ and the *in-neighborhood*, $N^-(v)$, of v is given by $\{u \mid (u, v) \in E\}$. The *out-degree*, $d^+(v)$, of a vertex v is given by $d^+(v) = |N^+(v)|$. Similarly, the *in-degree*, $d^-(v)$, of v is given by $|N^-(v)|$.

A digraph $G = (V, E)$ is *symmetric*, if for every edge $(u, v) \in E$ there is an edge $(v, u) \in E$. The *underlying graph* of a directed graph $G = (V, E)$ is the non-oriented graph $G' = (V', E')$ such that $V' = V$ and $E' = \{(u, v) \mid u \neq v$ and either $(u, v) \in E(G)$ or $(v, u) \in E\}$.

Let $G = (V, E)$ and $G' = (V', E')$ be two graphs. We say that G' is a *subgraph* G, if $V' \subseteq V$ and $E' \subseteq E$. If G' is a subgraph of G and $V' = V$, then we say that G' is a *spanning subgraph* of G. If V^* is a subset of V, then $G[V^*] = (V^*, E^*)$, where $E^* = \{(u, v) \mid u, v \in V^*$ and $(u, v) \in E\}$, is the subgraph of G *induced* by V^*. Similarly, if E^* is a subset of E, then $G[E^*] = (V^*, E^*)$, where $V^* = \{v \mid (u, v) \in E^*\}$, is the subgraph induced by the edges E^*.

A simple graph $G = (V, E)$ is *complete*, if $(u, v) \in E$ for all $u, v \in V$. For every $n \geq 1$, K_n denotes the complete graph with n vertices. A *stable set*, or *independent set*, is a subset V' of V such that there are no edges in $G[V']$.

A simple *path* $P = (v_1, e_1, v_2, e_2, \ldots, e_{p-1}, v_p)$, in a graph $G = (V, E)$, is an alternated sequence of vertices and edges of G with $p \geq 1$ such that $e_i = (v_i, v_{i+1})$, for all $i \in [1, p-1]$, and no vertices nor edges are repeated in P. The *length* of a path P is the number of its edges. When G is a simple graph, P can be determined by its vertices. The *extremities* of a path P are its first and last vertices and all its other vertices are called *internal*. A *cycle* is defined in a similar manner to a path, with the exception that its extremities are adjacent. We say that G contains a P_n (resp. C_n), if it contains a path (resp. cycle) of length n (resp. $n - 1$) as subgraph.

In a graph $G = (V, E)$, the distance, $\text{dist}(u, v)$, between two vertices u and v is the minimum length of a path between u and v. When there are no paths between u and v, then $\text{dist}(u, v) = \infty$. The *diameter* of a graph $G = (V, E)$ is given by $\max_{u,v \in V} \text{dist}(u, v)$.

The *girth* of a graph $G = (V, E)$ is the minimum length of a cycle in G.

Two vertices u and v of G are said to be *connected*, if there is a path between u and v. A graph G is said to be *connected*, if all pairs of vertices are connected, that is, if $\mathrm{diam}(G) < \infty$. A connected maximal subgraph of G is called a *component* of G.

To help the reader, more specific definitions will be given along the text as they become necessary.

Part I

Pursuit-Evasion Games and Graph Decompositions

Chapter 2

Pursuit-Evasion Games and Decompositions

In this chapter, we present some of the most important results concerning graph decompositions and pursuit-evasion games on graphs.

We start by giving an overview on some graph decompositions, their applications and the hardness of computing graph decompositions. Then, we explain the relationship between some pursuit-evasion games, known as graph searching games, and graph decompositions. We finish this chapter by surveying motivations and hardness of problems related to directed graph decompositions and pursuit-evasion games on directed graphs.

2.1 Graph Decompositions

As seen in Chapter 1, graph decompositions is a subject that has several of both algorithmic and theoretical applications. In this section, we give a brief survey on the computational complexity of computing graph decompositions for general graphs.

Tree/Path Decomposition

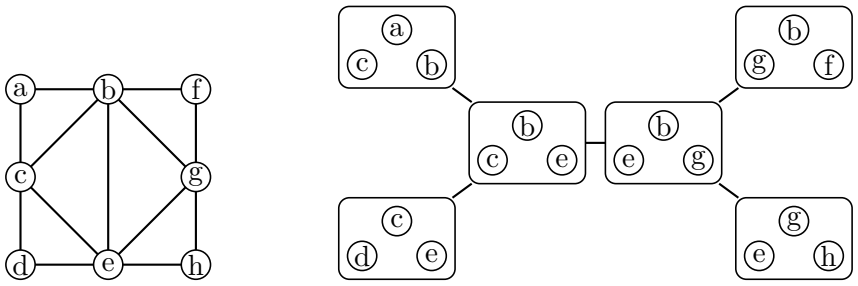
One of the most famous graph decomposition, due to its role in the *graph minors theory* developed by Robertson and Seymour [RS83, RS04] and its algorithmic applications, is the tree decomposition.

A *tree decomposition* $(T, \mathcal{X})$ of a graph $G = (V, E)$ is a tree T together with a family $\mathcal{X} = (X_t)_{t \in V(T)}$ of subsets (or *bags*) of V, such that:

1. $\bigcup_{t \in V(T)} X_t = V$,
2. for any edge $e = \{u, v\} \in E$, there is $t \in V(T)$ such that $u, v \in X_t$, and
3. for any $v \in V$, the vertices in $S = \{t \mid v \in X_t\}$ induce a subtree of T.

The width of $(T, \mathcal{X})$ is the value of $\max_{t \in V(T)}\{|X_t|\} - 1$ and the *tree width*, $\mathrm{tw}(G)$, of a graph G is the minimum width among all its tree decompositions. Figure 2.1 shows an example of a tree decomposition of a graph.

If T is restricted to be a path, we say that $(T, \mathcal{X})$ is a *path decomposition* of G, and the *path width*, $\mathrm{pw}(G)$, of G is the minimum width among its path decompositions. One



(a) Graph. (b) Tree Decomposition.

Figure 2.1: An example of tree decomposition of a graph. Vertices of the tree and its corresponding bag are represented by rectangles. Since each bag has size three, the width of this decomposition is two.

simple remark is that $\mathrm{tw}(G) \leq \mathrm{pw}(G)$, for any graph G, since any path decomposition of G is also a tree decomposition of G. On the other hand, in [Bod98], Bodlaender showed that $\mathrm{pw}(G) \leq \mathrm{tw}(G)O(\log n)$ for any graph G of order n.

One of the main reasons to study tree and path decompositions is that many problems in graph theory which are NP-complete in general become tractable when restricted to graphs with bounded tree width. Typically, these algorithms are based on a dynamic programming approach guided by a given tree decomposition of the input graph with running time that is polynomial in the size of the graph and, at least, exponential in the tree width of the given decomposition. Hence, if the family of graphs has tree width bounded by a constant, then these algorithms run in polynomial time for any graph of this family.

Most of these algorithms follow a similar pattern. In the case of algorithms guided by a tree decomposition, they are either given a tree decomposition $(T, \mathcal{X})$ of the graph as input or start by building a tree decomposition of the graph. Then, after choosing one vertex of the tree T as root, for each vertex v of the tree, let T_v be the subtree of T induced by v and its children. These algorithms proceed to compute a table, for each v in T, representing solutions to the subgraph induced by the vertices in any bag of $V(T_v)$. The solution of the problem, then, can be found by looking at the table of the root node.

By exploiting the fact that bags are separators of the input graph, the complexity of computing such tables are, often, polynomial (or even linear) in the number of vertices of the graph, but exponential on the width of the given tree decomposition or the tree decomposition constructed.

Thus, most of these algorithms are FPT where the parameter is either the tree width of the input graph, in the case the algorithm starts by computing the tree decomposition, or the tree width given as input to the algorithm. There are several problems that can be solved by this method such as the Hamiltonian circuit, the maximum independent set, the minimum dominating set problem or minimum vertex coloring problem [AP89].

In fact, a well celebrate result of Courcelle states that all problems which can be formulated in *Monadic Second Order Logic (MSOL)* can be solved in linear time on graphs with bounded tree width [Cou89].

Thus, an important challenge consists in computing tree decompositions of graphs that have small widths. Unfortunately, deciding if $\mathrm{tw}(G) \leq k$ [ACP87] and if $\mathrm{pw}(G) \leq k$ [OMK$^+$79] are NP-complete problems.

If, on the one hand, the problem of deciding if $\mathrm{tw}(G) \leq k$ or if $\mathrm{pw}(G) \leq k$ is hard,

on the other hand, in their seminal work on graph minors [RS83, RS04], Robertson and Seymour give a non-constructive proof of the existence of a $O(n^2)$ decision algorithm for the problems of deciding whether a graph belongs to some minor-closed class of graphs. An immediate consequence of this is the existence of polynomial time algorithms for deciding whether a graph has tree width or path width at most k, where k is a fixed parameter.

Bodlaender and Kloks [Bod96] proposed a FPT-algorithm to compute a tree decomposition or a path decomposition of a graph G with width $\mathrm{tw}(G)$ or $\mathrm{pw}(G)$ respectively. This algorithm is based on a dynamic programming approach from a tree decomposition of the input graph and runs in linear time on its number of vertices. However, its complexity is a function more than exponential[1] on the tree width of the input graph and the width of the given tree decomposition. Due to this more than exponential function, this algorithm is rather impractical even for very small values of k.

Therefore, great efforts have been made to design good approximation algorithms for computing tree decompositions of small width [BGHK95, Klo94]. In particular, Feige *et al.* proposes a polynomial time algorithm that constructs a tree decomposition of the input graph with width $O\left(\mathrm{tw}(G)\sqrt{\log \mathrm{tw}(G)}\right)$ [FHL05].

Special and q-branched Tree Decompositions

One of the consequences of the aforementioned result of Courcelle [Cou89] is that there are finite deterministic automatas for checking monadic second-order sentences on graphs. However, these automatas have size hyper-exponential on the tree width of the graph.

Then, in an attempt to reduce the size of these automatas, the *special tree decomposition* was introduced by Courcelle in [Cou10]. Special tree decompositions can be seen roughly as a mid-ground between path decompositions and tree decompositions.

Formally, the *special tree decomposition* $(T, \mathcal{X})$ of a graph $G = (V, E)$ is a rooted directed tree[2] T together with a family $\mathcal{X} = (X_t)_{t \in V(T)}$ of subsets (or *bags*) of V, such that:

1. $\bigcup_{t \in V(T)} X_t = V$,
2. for any edge $e = \{u, v\} \in E$, there is $t \in V(T)$ such that $u, v \in X_t$, and
3. for any $v \in V$, the vertices in $S = \{t \mid v \in X_t\}$ induce a directed path in T.

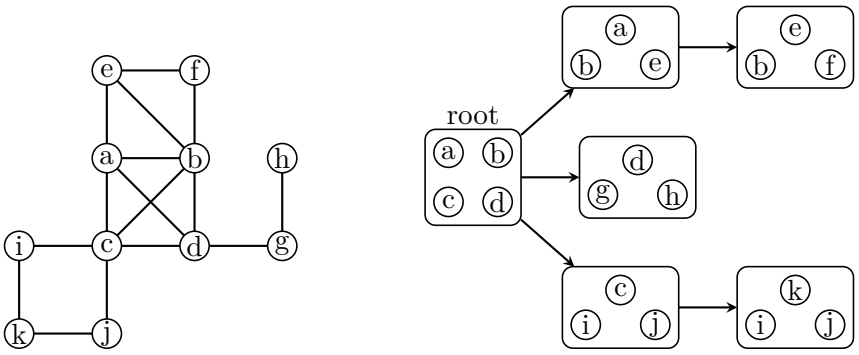
The width of $(T, \mathcal{X})$ is the value of $\max_{t \in V(T)}\{|X_t|\} - 1$ and the *special tree width*, $\mathrm{stw}(G)$, of a graph G is the minimum width among all its special tree decompositions. Figure 2.2 shows an example of a special tree decomposition of a graph.

A simple remark is that $\mathrm{tw}(G) \leq \mathrm{stw}(G) \leq \mathrm{pw}(G)$, since any path decomposition can be transformed into a special tree decomposition with same width, and any special tree decomposition can be transformed into a tree decomposition with the same width.

The problem of deciding if $\mathrm{stw}(G) \leq k$ is NP-complete, since, for any co-bipartite graph G, $\mathrm{pw}(G) = \mathrm{stw}(G) = \mathrm{tw}(G)$ [Mö96] and deciding if $\mathrm{tw}(G) \leq k$ is NP-complete [ACP87]. On the other hand, from the fact that the class of graphs with special tree width at most an integer k is minor-closed, there exists a FPT algorithm to compute $\mathrm{stw}(G)$ for any graph G, where the parameter is $\mathrm{stw}(G)$.

[1] A function $f(x)$ is more than exponential in x, if $f(x) \neq O(k^x)$ for any integer k.

[2] A rooted directed tree T is a rooted tree such that every arc is directed from the root to the leaves of T.



(a) Graph. (b) Special Tree Decomposition.

Figure 2.2: An example of a special tree decomposition of a graph. Vertices of the directed tree and its corresponding set are represented by rectangles. Since the largest bag of, the one of the root, has size four, the width of this decomposition is three.

Another variant of the tree decomposition, the *q-branched tree decomposition*, was introduced by Fomin *et al.* [FFN05]. This decomposition encompasses both path decompositions and tree decompositions as it will be latter explained in this section.

A node of a tree is said to be a *branching node* if it has degree at least three. A rooted tree T, with root r, is said to be q-branched if there are at most q branching nodes in each path between r and a leaf of T. A tree decomposition $(T, \mathcal{X})$ of a graph G is said to be q-branched if T is q-branched. Then, the q-branched tree width $\mathrm{tw}_q(G)$ of a graph G is the minimum width of all its q-branched tree decompositions.

The concept of q-branched tree decomposition encompasses both the concept of path decomposition and tree decompositions. Path decomposition are exactly the 0-branched tree decompositions, while tree decompositions are ∞-branched tree decompositions. Therefore, $\mathrm{tw}_0(G) = \mathrm{pw}(G)$ and $\mathrm{tw}_\infty(G) = \mathrm{tw}(G)$.

The hardness of deciding if a graph G has $\mathrm{tw}_q(G) \leq k$ is at least the same of deciding if a graph G has $\mathrm{tw}_0(G) \leq k$ or if $\mathrm{tw}_\infty(G) \leq k$, which are both NP-complete. However, for fixed $q \in \mathbb{N}^*$, it is unknown if this problem is NP-complete.

Similarly to the other parameters mentioned thus far, the class of graphs with q-branched tree width at most an integer k is minor-closed, hence there exists a FPT-algorithm for each $q \in \mathbb{N} \cup \{\infty\}$ for computing $\mathrm{tw}_q(G)$ where the parameter is $\mathrm{tw}_q(G)$.

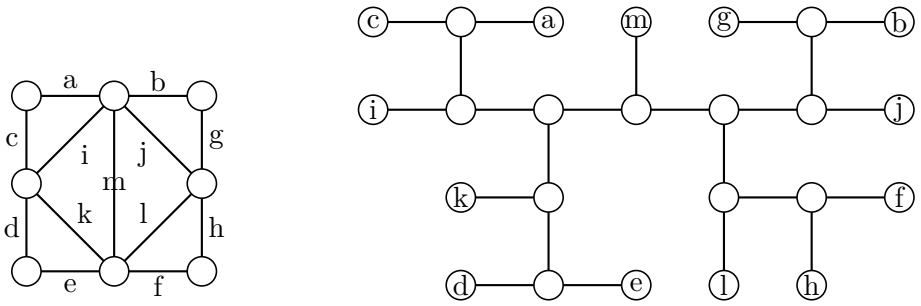
Albeit, the existence of FPT-algorithms for the decision problems related to the special tree width and the q-branched tree width is guaranteed, to the best of our knowledge there are no known explicit algorithms for that purpose.

Branch/Linear Decomposition

The notion of branch width has a close relationship to the one of tree width, since the branch width of a graph differs from its tree width by at most a multiplicative constant factor [BT97]. From the algorithmic standpoint, a branch decomposition also reflects some optimal tree structure arrangement of the graph it decomposes, hence it is possible to have algorithmic applications analogous to those of the tree decomposition.

A *branch decomposition* of a graph $G = (V, E)$ is a pair (T, σ), where T is a tree with vertices of degree at most 3 and σ is a bijection from the set of leaves of T to E. The width of an edge e in T is the number of vertices v in V such that there are leaves t_1 and t_2 in T which are in different components of $T[E(T) \setminus \{e\}]$ with $\sigma(t_1)$ and $\sigma(t_2)$ both

incident with v. The width of (T, σ) is given by the maximum width over all edges of T. Then, the *branch width*, $\text{bw}(G)$, of a graph G is the minimum width over all its branch decompositions. If $|E(G)| \leq 1$, the branch width of G is zero by definition. Figure 2.3 show an example of a branch decomposition of a graph.



(a) Graph (b) Branch Decomposition

Figure 2.3: A graph and its branch decomposition. The width of each edge in this branch decomposition is two, hence the width of this decomposition is two.

In order to define linear width, let $G = (V, E)$ be a graph with $|E| = m$. The *linear width*, $\text{lw}(G)$, of G is defined to be the least integer $k \geq 0$ such that the edges of G can be arranged in a linear ordering $(e_1, \ldots, e_m)$ in such a way that for every $i = 1, \ldots, m-1$, there are at most k vertices incident to edges that belong both to $(e_1, \ldots, e_i)$ and to $(e_{i+1}, \ldots, e_m)$.

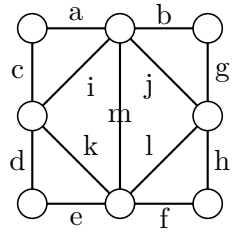
Linear orders over the edges of a graph and branch decompositions have a relationship that resembles the one between tree decompositions and path decompositions. A linear order $(e_1, \ldots, e_m)$ of the edges of a graph $G = (V, E)$, with $m = |E|$, can be described by a branch decomposition (T, σ) in the following manner. The tree T is obtained by starting with a path $P = (v_1, \ldots, v_m)$ and then, for each $i \in [1, m]$, we add a vertex v'_i and an edge between v_i and v'_i. Then, for each $i \in [1, m]$, map v'_i to e_i. Therefore, if $\text{lw}(G) \geq 2$ the linear width of a graph G is equal to the minimum width over all branch decompositions (T, σ) of G such that T is a caterpillar[3]. Figure 2.4 shows an example of a linear ordering over the edges of a graph represented by a branch decomposition.

As with tree decompositions, branch decompositions can be used as the basis of dynamic programming algorithms for many NP-hard optimization problems, one example being the traveling salesman problem [CS03]. Sometimes, the branch decomposition might work even better than the tree decomposition in the development of FPT-algorithms as argued in [FT06]. This happens because algorithms using the branch decomposition to solve a particular problem might have a better complexity which is partly based on the width of the branch decomposition than an algorithm, for this same problem, based on the tree decomposition.

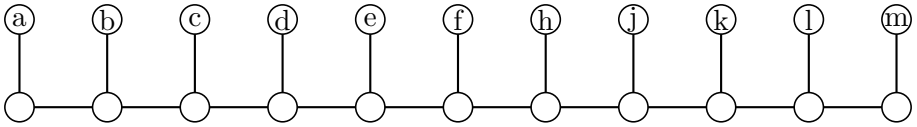
Therefore, it is important to efficiently construct branch and linear decompositions. Unfortunately, the problems of deciding if $\text{bw}(G) \leq k$ [ST94] and $\text{lw}(G) \leq k$ [Thi00] are NP-complete. However, for planar graphs, the branch width can be computed exactly in polynomial time [ST94], this in contrast with the tree width for planar graphs whose complexity is an open problem.

Fortunately, since the classes of graphs with branch width or linear width at most k, for any k, are minor closed, we also have that polynomial time algorithms exist for

[3]A caterpillar is a 1-branched tree.



(a) Graph.



(b) Branch Decomposition.

Figure 2.4: The linear decomposition, defined by the ordering $(a, b, c, d, e, f, g, i, j, k, l, m)$, of the graph in sub-figure (a) is represented by the branch decomposition shown in sub-figure (b).

deciding whether a graph has branch width or linear width at most k, where k is a fixed parameter. In fact, Bodlaender and Thilikos proposed linear time algorithms for deciding if the branch width [BT97] and the linear width [BT04] of a graph is at most a constant k. Moreover such algorithms successfully constructs a branch decomposition or a linear ordering with width at most k, in case they exist. Techniques used in these algorithms are based on the ones used in the algorithms for computing the tree and path decomposition of Bodlaender and Kloks in [BK96].

Carving/Cut Decomposition

The concepts of carving and cut decompositions are analogous to the concepts of branch and linear decompositions, when instead of mapping or ordering edges, we map or order vertices of the graph.

Formally, a *carving decomposition* of a graph $G = (V, E)$ is a pair (T, σ), where T is a tree with vertices of degree at most 3 and σ is a bijection from the set of leaves of T to V. The width of an edge e in T is the number of edges e' in E such that there are leaves t_1 and t_2 in T in different components of $T[E(T) \setminus \{e\}]$ with $\sigma(t_1)$ and $\sigma(t_2)$ both incident to e'. The width of (T, σ) is given by the maximum width over all edges of T. Then, the *carving width*, $\text{carw}(G)$, of a graph G is the minimum width over all its carving decompositions.

To formally define the cut width of a graph, let $G = (V, E)$ be a graph with $|V| = n$. The *cut width* $\text{cw}(G)$ of G is defined to be the minimum integer $k \geq 0$ such that the vertices of G can be arranged in a linear ordering $(v_1, \ldots, v_n)$ in such a way that for every $i = 1, \ldots, n-1$, there are at most k edges incident to vertices that belong both to $(v_1, \ldots, v_i)$ and to $(v_{i+1}, \ldots, v_m)$.

Cut width and carving width share the same relationship as the linear width and the branch width. That is, for every graph G such that $\text{cw}(G) \geq 2$, each linear order on the vertices of G can be represented by a carving decomposition (T, σ) of G, such that T is a caterpillar.

The decision problems related to cut width, commonly known as the Minimum Cut Linear Arrangement, and to the carving width are both NP-complete [MS88, ST94]. Fortunately, similarly to their counterparts, the classes of graphs with carving width or cut width at most k are minor-closed, which guarantees the existence of an FPT-algorithm

to decide if a graph has carving width or cut width at most k where k is the parameter. In fact, Bodlaender *et al.* [BST00] proposed an FPT-algorithm, with parameter k, that runs in linear time on the number of vertices of the input graph, to decide if the cut width or the carving width of said graph is at most k.

Relations Between Graph Widths

In this section we already mentioned relationships between some graph widths. Now, we further examine these relationships in Table 2.1.

Table 2.1: Table showing relationships among graph width parameters. These are true for any simple graph G and any $q \geq 0$.

Inequality			Reference
$\mathrm{tw}(G)$	$\leq$	$\mathrm{tw}_q(G)$	(By definition)
$\mathrm{tw}(G)$	$\leq$	$3\,\mathrm{bw}(G)/2$	[RS91]
$\mathrm{tw}(G)$	$\leq$	$\mathrm{stw}(G)$	[Cou10]
$\mathrm{tw}(G)$	$\leq$	$3\,\mathrm{carw}(G)$	[Thi00]
$\mathrm{tw}(G)$	$\leq$	$\mathrm{pw}(G)$	(By definition)
$\mathrm{pw}(G)$	$\leq$	$O(\mathrm{tw}(G)\log\lvert V(G)\rvert)$	[Bod98]
$\mathrm{pw}(G)$	$\leq$	$\mathrm{cw}(G)$	[Thi00]
$\mathrm{pw}(G)$	$\leq$	$\mathrm{lw}(G)$	[BT04]

Since the tree width of a graph is at most its path width, Table 2.1 also indicates that all the aforementioned widths are upper bounds for the tree width.

2.2 Graph Searching Games and Decompositions

In this section, we present some pursuit-evasion games known as graph searching games. These games have a close relationship with graph decompositions as it will be explained further.

One common characteristic among most graph searching games is that the game is played simultaneously by the two players. Meaning that each player may make its moves at any point during the game.

We start by exploring the *Node Search* game and its relationship with the tree and path decompositions.

Node Search, Monotonicity and Graph Decompositions

The *Node Search* (or *Helicopter Search*), defined by Seymour and Thomas [ST93], is one of the most famous graph searching games. This is mainly due to its close relationship to the tree decomposition as it will be further explained in this section.

In the *Node Search* game, the two players are the *cops* and the *robber*. The cops are tokens that stand at vertices of the graph while the *robber* is one token that also stands at vertices of the graph. Cops move by "boarding an helicopter" and flying from one vertex to another vertex of the graph. In other words, to move from one vertex to another vertex of the graph, a cop must remove itself from the graph for an amount of time that is not instantaneous. The robber can, at any time, move from its current position to another if there is a path between its current position and its destination that does not contain any cop. If the robber can move, its movement is considered instantaneous. The cops

win if a cop is able to land on the vertex the robber currently stands at and the robber cannot escape. In other words, the cops win the game if they are able to capture the robber. The robber wins, if it is able to avoid capture indefinitely. It is clear that n cops are sufficient to capture a robber in any graph of order n by occupying each vertex of the graph. Hence, the question is what is the minimum number of cops such that the cops can always guarantee the capture of the robber.

There are two main versions of the *Node Search*, depending on whether the cops have knowledge of the current position of the robber. In the *visible Node Search* the cops know the position of the robber at all times, while in the *invisible Node Search* the cops do not know the position of the robber, but when they capture it. Let $\mathrm{ns}_v(G)$, the *visible node search number*, and $\mathrm{ns}_i(G)$, the *invisible node search number*, be the minimum number of cops necessary to guarantee that the cops can always win against the robber in a graph G in the visible and invisible *Node Search* respectively. The first relation between graph searching games and graph decompositions is that, for any graph G, $\mathrm{ns}_v(G) \leq \mathrm{tw}(G)+1$ and $\mathrm{ns}_i(G) \leq \mathrm{pw}(G)+1$. The proofs for these inequalities are not very hard, since tree and path decompositions offer a natural way of searching a graph for the robber as can be seen in Figure 2.5.

A *strategy* for the cops is a sequence of movements, which may be based on the current position of the robber if the cops have such knowledge, that describes where each cop should move. A strategy is *winning*, if by following this strategy the cops are guaranteed to win the game against the robber. As defined in Chapter 1, strategies for the cops are *monotone*: if once a cop leaves a vertex, no other cop occupies this vertex for the remainder of the game. Equivalently, a strategy is *monotone*, if the area reachable by the fugitive never increases. One main characteristic of strategies designed by sequentially occupying bags of a tree/path decomposition is that these strategies are monotone as seen in Figure 2.5. Another main question in graph searching is if, by restricting the cops to play with only monotone strategies, we increase the number of cops necessary to capture the robber. If the answer is no, then we say that the graph searching game in question is *monotone*. Normally, monotonicity plays a major role in proving that a graph searching game is in NP, since a monotone strategy provides a certificate that can be checked in polynomial time. Another reason for the importance of monotonicity is that monotone games are, often, equivalent to some particular graph decompositions.

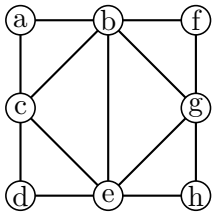
The *invisible Node Search* was first shown to be monotone in [KP86]. Moreover, by combining this with the results in [Kin92, Mö90], we have that, for any graph G, $\mathrm{ns}_i(G) = \mathrm{pw}(G)+1$. As consequence, we have that computing $\mathrm{ns}_i(G)$ is as hard as computing $\mathrm{pw}(G)$.

The *visible Node Search* is also monotone as it was shown by Seymour and Thomas in [ST93]. In fact, they prove the monotonicity by showing a sequence of equivalences that results in $\mathrm{ns}_v(G) = \mathrm{tw}(G)+1$ for any graph G, which also proves that the decision problem associated with $\mathrm{ns}_v(G)$ is NP-complete.

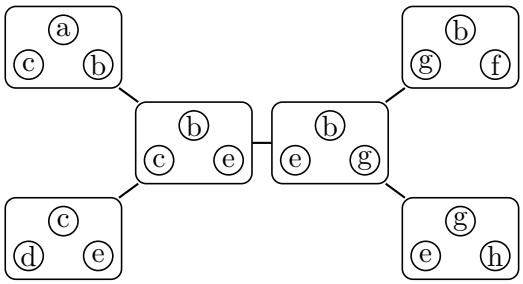
Other Searching Games and Graph Decompositions

There are several variants of graph searching games depending on conditions of capture, restrictions on the behaviors of players, whether the position of the robber is known for the cops, etc. These variants are mainly motivated by problems in practice or inspired by theoretical studies in Graph Theory such as graph decompositions.

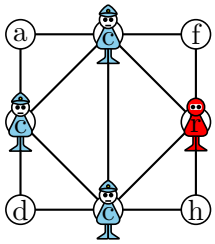
The first graph searching game, the *Edge Search* game, was introduced by Parsons



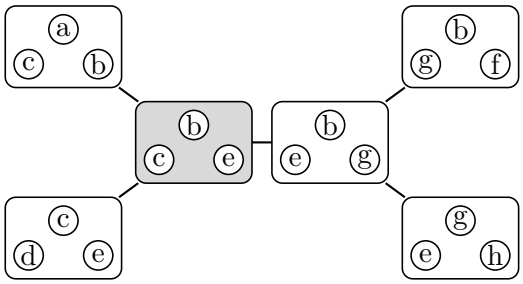
(a) Graph.



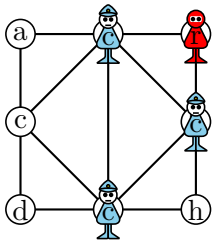
(b) A tree decomposition.



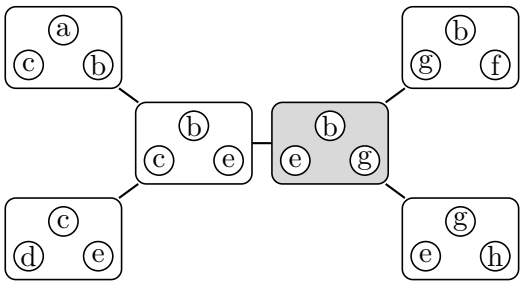
(c) The cops are placed in b, c and e. the robber moves to g



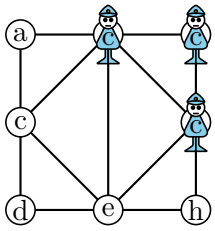
(d) The bag occupied by the cops is grayed out.



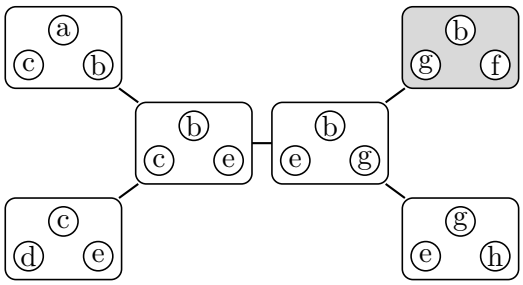
(e) The cop in c moves to g, The robber moves to f



(f) When moving from c to g, the vertices b and e separates the graph. Then, the robber must move to either f or g.



(g) The cop in e moves to f, The robber cannot flee from f and its captured



(h) When moving from e to f, the robber cannot escape.

Figure 2.5: This scheme shows how a robber in the Node Search game can be captured with the aid of a tree decomposition of the graph.

[Par78]. This game was motivated by the problem of finding a spelunker[4] who is lost and wandering unpredictably in a system of caves. The *Edge Search* game differs from the *Node Search* game by three factors: the placement of the robber, the rules for the movements of the cops and the condition of capture of the robber. The robber is able

[4]A spelunker is someone who makes a hobby of exploring and studying caves.

to stand either on vertices or edges of the graph. The cops along with being able to be placed on a vertex or removed from a vertex can also slide through an edge of the graph. That is if $e = (u, v)$ is an edge and there is a cop currently standing at u, then it can move through edge e to v. The robber is captured, if it cannot escape a cop landing or sliding through the position it currently stands.

Let the *invisible edge search number*, $\mathrm{es}_i(G)$, be the minimum number of cops necessary to guarantee the capture of an invisible robber in the *Edge Search* game on a graph G. One natural question that arises is if there is any relationship between the values $\mathrm{es}_i(G)$ and $\mathrm{ns}_i(G)$.

An answer to this question is due to Bienstok and Seymour [SB91]. They showed that, for any graph G, $\mathrm{ns}_i(G) - 1 \leq \mathrm{es}_i(G) \leq \mathrm{ns}_i(G) + 1$. Roughly, the reason behind this is that the rule of sliding through an edge (u, v) can be interchanged by adding an extra cop at v and then removing a cop from u. Examples showing that these inequalities are tight and can be found in complete graphs. For any complete graph of order $n \geq 2$, $\mathrm{ns}_i(G) = n$ and $\mathrm{es}_i(G) = n - 1$.

The first proof that *invisible Edge Search* is monotone is due to Lapaugh [LaP93]. Then, in [SB91], Bienstok and Seymour proposed a method that gives a succinct proof for the monotonicity of the *invisible Edge Search*. In order to show the monotonicity of the *invisible Edge Search*, they first introduced another variant of the graph searching problem, the *Mixed Search*. Then, they showed that the invisible mixed search is monotone, which implies the monotonicity of both the *invisible Edge Search* and the *invisible Node Search*.

The *Mixed Search* differs from the *Edge Search* only in the capture conditions of the robber. In the *Mixed Search*, the robber can be captured either if a cop lands on the vertex where the robber currently stands, if there are cops in both endpoints of the edge currently occupied by the robber, or if a cop slides through the edge currently occupied by the robber and it can not escape. That is, the robber is captured in the *Mixed Search* whenever it is captured in the Node Search game, in the Edge Search game, or if it stands on an edge while both its endpoints are occupied by cops. Let *invisible mixed search number*, $\mathrm{ms}_i(G)$, be the minimum number of cops necessary to guarantee the capture of an invisible robber in the *Mixed Search* game on a graph G.

Invisible mixed search has a close relationship with the invisible node search and the invisible edge search. In [SB91], it was shown that $\mathrm{ms}_i(G) \leq \mathrm{es}_i(G) \leq \mathrm{ms}_i(G) + 1$ and that $\mathrm{ms}_i(G) \leq \mathrm{ns}_i(G) \leq \mathrm{ms}_i(G) + 1$.

The edge search numbers and the mixed search numbers have a close relationship to the path width and tree width, due to their correlation with the node search numbers. However, there are graph decompositions which are, sometimes, more closely related with these parameters.

In particular, for every graph G, the invisible mixed search number of G was shown to be "equivalent" to the proper path width of G [TUK95]. A path decomposition $(P, \mathcal{X})$ of G is said to be proper if, for every X_i, X_j, X_k in $\mathcal{X}$, none of which are subsets of the other. Therefore, the *proper path width* $\mathrm{ppw}(G)$ of a graph G is the minimum width over all proper path decompositions of G. Then, $\mathrm{ms}_i(G) = \mathrm{ppw}(G) + 1$ [TUK95].

In the case of the Edge Search, Makedon and Sudborough showed a relationship between the cut width of a graph and its invisible edge search number. More precisely, they showed that, for any graph G with maximum degree Δ, $\mathrm{es}_i(G) \leq \mathrm{cw}(G) \leq \lfloor \Delta/2 \rfloor (\mathrm{es}_i(G) - 1) + 1$ [MS89]. This implies that for any graph G with maximum degree 3 we have that $\mathrm{es}_i(G) = \mathrm{cw}(G)$.

The discussion up to now, implies that graph searching games are monotone in general. Is it true that requiring the cops to play in a monotone manner does not increase the number of cops for all graph searching games? The answer to this question is no for both the *connected visible Node Search* [FN08] and the *connected invisible Edge Search* [YDA09]. A connected graph searching game restricts the cops to play in such a way that the subgraph not reachable by the robber is connected in every step of the game. While the connected visible or invisible Node Search are not monotone, in [BFST03], it was shown that the *internal connected invisible Edge Search* is monotone. An *Edge Search* game is said to be internal, if cops cannot be removed from a vertex. That is, once a cop first occupies any vertex it can only move to other vertices by "sliding" through edges of the graph. A cop cannot move by "boarding an helicopter" and latter "descending" on another vertex, it must move through the edges of the graph.

In [YDA09], Yang *et al.* investigated the cost of monotonicity in the *connected invisible Edge Search*. They show that restricting the cops to play monotonously can increase the number of cops by an arbitrary amount. Then, Fraigniaud and Nisse showed an infinite family of graphs where restricting monotonicity on the *connected visible Edge Search* increases the number of cops by one [FN08].

Often, one of the open questions regarding graph searching games that are not monotone is if the associated decision problems are in NP. Since winning strategies that are not monotone can take an arbitrary large, albeit finite, number of steps in order to capture the robber, these strategies cannot be used as a certificate to show that the graph searching problem in question is in NP.

Due to their close relationship to graph decompositions, graph searching games can be considered as another approach into designing powerful graph decompositions. However, all decompositions and graph searching games presented up to this point are concerned only with undirected graphs. Then, one natural question that arises is how can these concepts be extended to directed graphs.

In the next section we explore the challenges associated with extending the notion of graph decompositions and graph searching games to directed graphs.

2.3 Directed Graph Decompositions and Directed Graph Searching

During the last few years, an important research effort has been done in order to design graph decompositions for directed graphs that are as powerful as the path or the tree decompositions are for undirected graphs. A graph decomposition, to be considered powerful, should have two main properties: (1) be algorithmically useful and (2) have nice structural properties such as being closed under taking subdigraphs and some form of arc contractions [GHK$^+$10]. Because graph searching games are equivalent to path and tree decompositions in undirected graphs, several attempts have been done to define such games in directed graphs [Bar06, HK08, BDH$^+$12].

In this section we explore some well known directed graph decompositions and their relationship with graph searching games in directed graphs.

Directed Tree Decomposition

One of the first directed graph decompositions, the *directed tree decomposition*, was introduced by Johnson *et al.* in [JRST01].

An *arborescence* T is a rooted directed tree. Therefore, there is a directed path from the root of T to any leaf of T. For any r and r' in $V(T)$, we say that $r' \geq r$ if there exists a directed walk in T with initial vertex r and terminal vertex r' and we say that $r' > r$ if $r' \geq r$ and $r' \neq r$. Similarly, for all $e = (u, r) \in E(T)$, we say that $r' \geq e$ if $r' \geq r$.

Let D be a digraph and $Z \subseteq V(D)$. We say that a set $S \subseteq V(D) \setminus Z$ is *Z-normal* if there is no directed walk in $D \setminus Z$ with first and last vertex in S and with internal vertex in $D \setminus (Z \cup S)$. That is, a set S is Z-normal if every path that leaves S and goes back to S must pass through a vertex in Z.

A *directed tree decomposition* of a directed graph $D = (V, E)$ is a triple $(T, \mathcal{X}, \mathcal{W})$ where T is an arborescence $\mathcal{X} = \{X_e \mid e \in E(T)\}$ and $\mathcal{W} = \{W_v \mid v \in V(T)\}$ are families of subsets of vertices of $V(D)$ such that:

- $\mathcal{W}$ is a partition of $V(D)$ into non empty sets, and
- if $e \in E(T)$, then $\bigcup\{W_v \mid v \in V(T), v \geq e\}$ is X_e-normal.

The width of $(T, \mathcal{X}, \mathcal{W})$ is $\max_{v \in V(T)} |W_v \cup \bigcup\{X_e \mid e \text{ incident to } v\}|$. Then, the *directed tree width*, $\mathrm{dtw}(D)$, of a directed graph D is given by the minimum width over all its directed tree decompositions.

Directed tree decompositions have a close relationship with tree decompositions. The directed tree width of a digraph is equal to the tree width of its underlying graph [JRST01].

The directed graph searching game associated with this decomposition is the *Strongly Connected Components (SCC) Search*. In this game, the cops play as in the Node search game for undirected graphs. In other words, each cop can be either placed on a vertex or removed from a vertex of the graph. The robber, however, is restricted to move through directed paths, but can only move if there is also a directed path from its intended destination back to its current position that is free of cops.

By following the same reasoning for building Node Search strategies from tree decompositions, a directed tree decomposition $(T, \mathcal{X}, \mathcal{W})$ can be used to build a winning strategy to capture a *visible* robber in the SCC Search game using a number of cops equal to the width of the directed tree decomposition. That is, cops are first placed onto $W_r \cup \bigcup_{e \text{ incident to } r} X_e$, where r is the root of T. Then, let v be the vertex of T such that there are cops on every vertex of $W_v \cup \bigcup_{e \text{ incident to } v} X_e$, the cops move from $W_v \cup \bigcup_{e \text{ incident to } v} X_e$ to $W_{v'} \cup \bigcup_{e' \text{ incident to } v'} X_{e'}$ where v' is the child of v such that the robber is in a vertex of W_u, $u \geq v$ until the robber is captured.

Hence, the directed tree width of a graph upper bounds the number of necessary cops to capture a *visible* robber in the SCC Search game. Albeit, strategies designed in this manner might not be monotone, meaning that cops might reoccupy vertices of the graph, they are "robber" monotone, meaning that the area reachable by the robber never increases. In [JRST01], it was shown that enforcing "robber" monotonicity might increase the number of cops to capture a visible robber by a multiplicative factor of three. While the previous result does not guarantee the "robber" monotonicity of the game, it implies that the cost of requiring monotonicity is linear in the number of cops. It was unknown whether the visible SCC Search was "robber" monotone until Adler showed, in [Adl07], that this game is not monotone by providing an example where 4 cops have a winning strategy, but 4 cops do not have a "robber" monotone winning strategy.

DAG Decomposition

The restriction that the robber has a returning path free of cops to move in the *SCC Search* game is rather unnatural. For this reason, Berwanger *et al.* proposed the *visible Directed Node Search* game which is the SCC Search where the robber does not have this restriction [BDH$^+$12]. In other words, in the *visible Directed Node Search* the robber can move if it has a directed path free of cops to its intended destination. They also propose a graph decomposition, the *DAG decomposition* that is closely related to monotone strategies in the *visible Directed Node Search*.

In order to introduce the DAG decomposition we need to first state some definitions. Let G be a directed graph and $W, Y \subseteq V(G)$. We say that W *guards* Y if for all $(u, v) \in E(G)$ with $u \in Y$ we have that $v \in W \cup Y$. If $D = (V, E)$ is a DAG, then let $\preceq_D$ be the partial order over V obtained by the reflexive transitive closure of E. That is, if there is a directed path from a vertex u to a vertex v in D then $u \preceq_D v$.

The *DAG decomposition* of a directed graph G is a pair $(D, \mathcal{X})$ where D is a DAG and $\mathcal{X} = \{X_v \mid v \in V(D)\}$ is a family of subsets of $V(G)$ such that:

- $\bigcup_{v \in V(D)} X_v = V(G)$,
- for all vertices $d, d', d'' \in V(D)$ such that $d \preceq_D d' \preceq_D d''$ we have that $X_d \cap X_{d''} \subseteq X_{d'}$,
- for all edges $(d, d') \in E(D)$ the set $X_d \cap X_{d'}$ guards $\bigcup_{d' \preceq_D d''} X_{d''}$, and
- for any source $d \in V(D)$ we have that $\bigcup_{d \preceq_D d'} X_{d'}$ is guarded by $\emptyset$.

The width of $(D, \mathcal{X})$ is $\max_{d \in V(D)} |X_d|$ and the *DAG width*, $\mathrm{dagw}(G)$, of G is the minimum width over all DAG decompositions of G.

In [BDH$^+$12], it was shown that monotone strategies for the cops in the *visible Directed Node Search* game have a close relationship with *DAG decompositions*. Meaning that the minimum number of cops to guarantee the capture of the robber in a monotone way in this game is equal to the DAG width of the graph in which it is played. Similarly with the *visible SCC Search*, the *visible Directed Node Search* is not monotone [KO08]. The family of graphs presented by Kreutzer and Ordyniak show that the cost of requiring monotonicity is at least a multiplicative factor. However, unlike the *visible SCC Search*, a non trivial upper bound on the cost of requiring monotonicity is still open.

From the fact that the robber is more powerful in the *visible Directed Node Search* compared to the *visible SCC Search*, it is easy to see that the number of cops necessary to guarantee the capture of the robber in the former is at least as big as in the latter. Hence, $\mathrm{dtw}(G) \leq \mathrm{dagw}(G)$ for any directed graph G. Then, a natural question is how big can this gap be? In [BDH$^+$12], Berwanger *et al.* proposed a family of graphs such that the DAG width of any graph in this family can be arbitrarily large, while their directed tree width is one. Moreover, they also show that if a directed graph G is such that $\mathrm{dagw}(G) \leq k$ then $\mathrm{dtw}(G) \leq 3k + 1$. Proving that having a small DAG width implies in having a small directed tree width.

Kelly Decomposition

A robber in a graph searching game is said to be inert if it is only able to move immediately before a cop moves to the vertex or through the edge it is currently occupying. In other words, the robber is lazy, only moving when it is in immediate danger of being captured.

In undirected graph searching games, the *visible Node Search* is equivalent to the *inert invisible Node Search* [DKT97]. That is, the same number of cops are necessary to capture a visible robber or an invisible and inert robber in the *Node Search*. In [HK08], Hunter and Kreutzer proposed the *inert invisible Directed Node Search*, which follows the same rules as the *visible Directed Node Search* with the exception that the robber is invisible and inert. Then, a natural question that arises is the equivalence between the *visible Directed Node Search* and *inert invisible Directed Node Search* in the same manner as the one between *visible Node Search* and *inert invisible Node Search*.

In [HK08], it was shown that these two games are not equivalent. More precisely, for all $k \geq 1$, there are graphs where $4k$ cops are necessary to capture a robber in the *visible Directed Node Search*, while only $3k$ cops are necessary to capture a robber in the *inert invisible Directed Node Search*. This is true even if the cops are restricted to capture the robber using a "robber" monotone strategy in the *inert invisible Directed Node Search*.

Dendris *et al.* also proposed a directed graph decomposition, the *Kelly decomposition*, related with monotone strategies for the *inert invisible Directed Node Search*.

The *Kelly decomposition* of a directed graph G is a triple $(D, \mathcal{X}, \mathcal{W})$ where D is a DAG, $\mathcal{X} = \{X_d \mid d \in V(D)\}$ and $\mathcal{W} = \{W_d \mid d \in V(D)\}$ are a family of subsets of $V(G)$ such that:

- $\mathcal{X}$ partitions $V(G)$,
- for all $d \in V(D)$, W_d guards $\bigcup_{d \preceq_D d'} X_{d'}$, and
- for all $d \in V(D)$ there is a linear order on its children $d_1, \ldots, d_p$ such that for all $1 \leq i \leq p$, $W_{d_i} \subseteq X_d \cup W_d \cup \bigcup_{j<i}(\bigcup_{d_j \preceq_D u} X_u)$. For all $d \in V(D)$ such that d is a source, there is a linear order on its children $d_1, \ldots, d_p$ such that for all $1 \leq i \leq p$, $\bigcup_{j<i}(\bigcup_{d_j \preceq_D u} X_u)$.

The width of $(D, \mathcal{X}, \mathcal{W})$ is $\max_{d \in V(D)} |X_d \cup W_d|$ and the *Kelly width*, $\text{kelly}(G)$, of G is the minimum width over all Kelly decompositions of G.

In the same manner as DAG decompositions of G can be used to design monotone winning strategies for the cops in the *visible Directed Node Search*, Kelly decompositions can be used to design monotone winning strategies for the cops in the *inert invisible Directed Node Search*.

Motivated by the equivalence between the inert invisible robber and the visible robber in the undirected *Node search*, we might wonder about the relationship between the Kelly width and the DAG width of a directed graph. In [HK08], it was also shown that, for any directed graph G, if $\text{kelly}(G) = 1$ or $\text{kelly}(G) = 2$, then $\text{kelly}(G) = \text{dagw}(G)$. However, from the fact that the *SCC Search* and *inert invisible Directed Node Search* are not equivalent, we have that there are graphs such that $\text{kelly}(G) = (3\,\text{dagw}(G))/4$. It is still an open question whether DAG width and Kelly width can be bounded within a constant factor of the other.

In the following, we study the relationship between the directed graph searching game where the robber is still invisible but not inert any more with directed graph decompositions.

Directed Path Decomposition

The *directed path decomposition*, proposed by Barát in [Bar06], is closely related with the *invisible Directed Node Search*.

A *directed path decomposition* of a directed graph G is a sequence $\mathcal{W} = (W_1, \ldots, W_n)$ of subset of vertices of $V(G)$ such that:

- $\bigcup_{1 \leq i \leq n} W_i = V(G)$,
- if $i \leq j \leq k$ then $W_i \cap W_k \subseteq W_j$, and
- for all $(u, v) \in E(G)$ we have that there exists $1 \leq i \leq j \leq n$ such that $u \in W_i$ and $v \in W_j$.

The width of a path decomposition is given by $\max_{1 \leq i \leq n} |W_i| - 1$. Then, the *directed path width*, $\mathrm{dpw}(G)$, of G is given by the minimum width over all its directed path decompositions.

Unlike its visible version, the *invisible Directed Node Search* was "almost" shown to be monotone in [Bar06]. To show this, Bárat used a similar method as the one used by Bienstok and Seymour to show that the *invisible Edge Search* is monotone. That is, he defined the *invisible Directed Mixed Search* and by showing the monotonicity of the *invisible Directed Mixed Search*, Bárat was able to prove the monotonicity of the *invisible Directed Edge Search* and that the cost of enforcing monotonicity of the *invisible Directed Node Search* might be of one extra cop. Then, in [Hun06], Hunter improved this result showing that the *invisible Directed Node Search* is, in fact, monotone. Moreover, this means that the minimum number of cops to capture the robber in the *invisible Directed Node Search* of a graph is equal to its directed path width plus one.

The directed path decomposition also has a relationship with the DAG decomposition similar to the one of tree decompositions and path decompositions. That is, directed path decompositions are DAG decompositions where the DAG is a directed path [BDH$^+$12]. Hence, for any directed graph G, we have $\mathrm{dagw}(G) \leq \mathrm{dpw}(G)$. Moreover, this gap can be arbitrarily large since, in [BDH$^+$12], a family of graphs with DAG width two and arbitrarily large directed path width was presented. The family presented is the family of symmetric directed graphs such that their underlying graph are complete ternary trees. This means that, for any graph G in this family $\mathrm{dpw}(G) \leq O(\mathrm{dagw}(G) \log |V(G)|)$. However, it is unknown if this is true for every graph.

2.4 Objectives

As was stated in the beginning of this section, directed graph decompositions were proposed in an attempt to bring powerful results of graph decompositions from undirected graphs to directed graphs. Since graph decompositions can often be seen as particular pursuit-evasion games, namely graph searching games, we can approach the problem of designing powerful directed graph decompositions by studying directed graph searching games.

For these reasons, the next chapter is dedicated to the study of the monotonicity of the *Process game* and the design of a directed graph decomposition related to this game.

Another goal is to investigate the problem of computing graph decompositions. In Chapter 4, we propose a unified FPT-algorithm that can be used to compute any of the aforementioned widths. This algorithm is based on the representation of these decompositions with partitioning trees and a dynamic programming approach based on an efficient representation of these partitioning trees.

This is the first FPT-algorithm for the special tree width and q-branched tree width. Moreover, the proposed algorithm is not restricted to compute only the widths of the

aforementioned graph decompositions, it can compute any width measure of a set that follows some restrictions further explained on Chapter 4.

Chapter 3

Monotonicity of The Process Game

In this chapter we study the monotonicity property of a graph searching game, known as the *Process game*, which is played on directed graphs. This graph searching game has been defined in the context of the problem of routing reconfiguration in WDM networks [CPPS05].

We start the next section, Section 3.1, by briefly explaining the relationship between the routing reconfiguration problem and the *Process game*. Then, we proceed to show some known relationships between the *Process game* and other graph parameters. Section 3.2 is dedicated to the main result in this chapter, which states that the *Process game* is monotone. We propose a new directed graph decomposition, the *Process Decomposition*, and show an equivalence between this decomposition and the *Process game* in Section 3.3. Finally, in Section 3.4, we finish by proposing some future directions of research in this area.

3.1 Process Game and Routing Reconfiguration

The *Process game* and the *routing reconfiguration* problem have a very close relationship that is further explained in this section. We start this section by exploring this relationship. Roughly, solving any instance of routing reconfiguration problem is equivalent to solving the corresponding instance in the *Process game*.

In telecommunication networks such as wavelength division multiplexed (WDM) networks, due to the need of maintenance operations or simply a link failure, connections that are using these links must be rerouted. However, it might not be a simple task to reroute such connections, since other links on the network might be already at their full capacity. The goal of the routing reconfiguration problem, defined by Jose and Somani in [JS03], is to achieve such rerouting minimizing some criteria.

More formally, an instance of the *routing reconfiguration* problem, (N, C, I, F), is defined by a network N, a set of connections C, an initial routing I and a final routing F. The network is represented by a directed graph N. The set of connections is given by $C \subseteq V(N) \times V(N)$. An initial routing of these connections, I, is given by a set of directed paths in N joining each pair $(u, v) \in C$, with the restriction that two different paths do not share an arc of N. That is, these paths are arc-wise disjoint. The final

routing, F, is represented in the same manner as the initial routing. That is, the final routing, F is a set of arc-wise disjoint paths of N joining each pair $(u, v) \in C$.

Let P_a^i be the path joining two vertices of a connection $a \in C$ in its initial routing I, and P_a^f be the path joining two vertices of a connection a in its final routing F. The objective of the routing reconfiguration problem is to change the routing of the connections from the initial routing, I, to the final routing, F, while minimizing some criteria. In order to do this, we are allowed to apply some operations, that are explained below, on the current routing of the network. However two different connections, after applying any of these operations, cannot share a same arc on the network. That is, the paths defining the connections on the current routing are all arc-wise disjoint at any point during the rerouting. In the following, we describe each of the allowed operations.

Interrupt: when applied to a connection a, the result of this operation is that a is interrupted. That is we remove the path $P_a^i \in I$ joining the two nodes of a from the current routing on the network.

Re-establish: when applied to an *interrupted* connection a, it re-established the connection on its final routing. That is, we add the path $P_a^f \in F$ to the current routing on the network.

Switch: when applied to a connection a, the connection a is switched, instantaneously, from its initial routing to its final routing. That is, we remove the path $P_a^i \in I$ joining the two nodes of a from the current routing on the network, whilst adding the path $P_a^i \in F$ to the current routing.

Note that it is always possible to change from the initial routing to the final routing by interrupting every connection and then re-establishing these connections in their final routing.

There are several criteria that can be used to measure how "good" is a sequence of operations used to change the initial routing of the network into the final routing. For example, the total number of interruptions was studied in [CCM+11] and the maximum number of simultaneous interruptions during the rerouting was studied in [CPPS05].

Inspired by the routing reconfiguration problem, Coudert *et al.* introduced the *Process game* in [CPPS05]. In the *Process game*, a team of searchers aims at *processing* all nodes of a digraph. A node is said *safe* if all its out-neighbors are either occupied or already processed. Given a digraph $D = (V, A)$ where initially all nodes are unoccupied and not processed, a *monotone process strategy* is a sequence $(s_1, \ldots, s_n)$ of steps that results in processing all nodes of D, where each step s_i is one of the following three moves.

M_1**:** place a searcher (or agent) at node $v \in V$;

M_2**:** process a safe *unoccupied* node $v \in V$;

M_3**:** process a safe *occupied* vertex $v \in V$ and remove the searcher from it.

Note that, once a vertex is processed, it never becomes "unprocessed" at a latter stage of the game. Hence, if X_i is the set of vertices that are processed after step i, then $X_i \subseteq X_{i+1}$ for all $1 \leq i \leq n-1$. For this reason, we say that this is a *monotone process strategy*. The minimum number of searchers such that there exists a monotone process strategy for D is the *monotone process number*, denoted by $\text{monpr}(D)$.

The relationship between the *Process game* and the *routing reconfiguration* problem is mainly due to the *dependency digraph* defined by [JS03]. The dependency digraph

$D = (V, A)$ of an instance of the routing reconfiguration problem (N, C, I, F) has one vertex for each connection in the routing reconfiguration instance and there is an arc $e = (u, v) \in A$, if the connection given by u in its final routing shares an arc in the network with the connection given by v in its initial routing. That is, $V = C$ and $E = \{(u, v) \mid E(P_u^f) \cap E(P_v^i) \neq \emptyset\}$.

As observed in [CPPS05] a solution for the *routing reconfiguration* problem is equivalent to a solution to the *Process game* played on its dependency digraph. Here we roughly sketch how this is achieved. Consider an instance of the routing reconfiguration problem (N, C, I, F) and its dependency digraph D. Then, whenever a connection $a \in C$ is interrupted we use M_1 in $a \in V(D)$ and vice versa. That is, $a \in C$ is interrupted if, and only if, there is an agent on $a \in V(D)$. Whenever a connection $a \in C$ is re-established we use M_3 in $a \in V(D)$ and vice versa. That is, $a \in C$ is re-established if, and only if, the agent on $a \in V(D)$ is removed and $a \in V(D)$ is processed. Finally, whenever $a \in C$ is switched we use M_2 in $a \in V(D)$ and vice versa. That is, $a \in C$ is switched if, and only if, $a \in V(D)$ is processed without having an agent on it. Therefore, during a routing reconfiguration or during the processing of D, a connection $a \in C$ is on its final routing if, and only if, the vertex $a \in V(D)$ is processed. Consequently, the final routing is achieved on the rerouting problem if, and only if, all vertices of D are processed. Moreover, a connection $a \in C$ is interrupted if, and only if, the vertex $a \in V(D)$ is occupied by an agent.

Therefore, any monotone strategy for the *Process game* gives a corresponding strategy for the routing reconfiguration problem, in which the number of interrupted connections in any step is given by the number of occupied vertices in the *Process game*.

An important result is that, for any directed graph D, there is an instance of the reconfiguration problem such that D is its dependency digraph [CCM+11]. This means that we can focus solely on the *Process game* in order to understand both problems.

Process Game and Other Parameters

While the process number of digraphs has been mainly studied for its applications in the rerouting problem in WDM networks [CHM+09, Sol09, SP09], it is also related with some other graph parameters.

In [CPPS05], Coudert *et al.* showed that $\text{pw}(\bar{D}) \leq \text{monpr}(D) \leq \text{pw}(\bar{D}) + 1$, for any symmetric digraph[1] D, where $\bar{D}$ is the underlying graph of D. Moreover, the decision problem associated with the process number is NP-complete in general, but the process number can be computed in polynomial time in the class of (di)graphs D with $\text{monpr}(D) \leq 2$ [CS11] and in the class of trees [CHM12].

In [CCM+11], it was shown that the minimum feedback vertex set[2] is an upper bound for the process number of a directed graph. However, it is true that there are graphs with process number two and arbitrarily large minimum feedback vertex set.

Note also that, in undirected graphs (seen as symmetric digraphs), the monotone processing game is equivalent to the graph searching game, where an invisible robber is captured if all the neighbors of its position are occupied, i.e., it is not required that a cop occupies the same node as the robber, only its neighborhood. It is important to notice that the *Process game*, when played on a symmetric digraph is not equivalent to the invisible graph searching game. Given a symmetric directed graph D, the following

[1] A digraph $D = (V, A)$ is *symmetric* if, for any $(a, b) \in A$, then $(b, a) \in A$.

[2] A feedback vertex set is a set of vertices such that the result of their exclusion from the graph is a DAG.

inequation is true: $\mathrm{ns}_i(\bar{D}) - 1 \leq \mathrm{monpr}(D) \leq \mathrm{ns}_i(\bar{D})$, where $\bar{D}$ denotes the underlying graph of D and $\mathrm{ns}_i(\bar{D})$ denotes the minimum number of cops necessary to capture an invisible robber in $\bar{D}$. Moreover, these inequality are tight. Let K_n be the complete directed graph with n vertices, then $\mathrm{ns}_i(\bar{K_n}) = \mathrm{monpr}(K_n) + 1$ and, for every directed graph D, let D' be the directed graph obtained from D by adding a loop to every vertex of D, then $\mathrm{ns}_i(\bar{D'}) = \mathrm{monpr}(D')$.

In the case that D is a directed graph that is not symmetric, the *Process game* is equivalent to capturing a robber in the *Node Search* that moves in the opposite direction of the arc and must always move with a monotone strategy, otherwise it also loses the game. That is, if at any point the robber is captured it loses the game, while also losing if it cannot move any more. For example, the robber loses on any DAG D even against zero cops, since, at some point, the robber must move to a source of D being unable to move afterwards. This equivalence is easy to see by considering processed and occupied vertices in the *Process game* as vertices that are unreachable by the robber in the aforementioned graph searching game. Let $\mathrm{dns}_i(D)$ be the number of cops necessary to capture the robber in the *invisible Directed Node Search* of a directed graph D. Then, for any directed graph D, we have that $\mathrm{dns}_i(D) - 1 \leq \mathrm{monpr}(D) \leq \mathrm{dns}_i(D)$. Again the same examples above are sufficient to show that this inequality is tight in both sides.

Our Results

In order to study if the *Process game* is monotone when seen as the graph searching game described above, we consider the more general variant of non necessarily monotone processing game in this chapter.

That is, we allow processed nodes to become unprocessed. More precisely, a *process strategy* for a digraph D is a sequence $(s_1, \ldots, s_n)$ of steps that results in having all nodes of D in the processed state, where each step s_i consists of a move M_1 or M_2 or

M_3'**:** process an *occupied* vertex $v \in V$ and remove the searcher from it. If v was not safe then recontamination occurs. That is, successively, all processed vertices (including v) that have an unoccupied and unprocessed out-neighbor become unprocessed.

The minimum number of searchers such that there exists a process strategy for D is the *process number*, denoted by $\mathrm{pr}(D)$.

In [CHM12], it was proved that $\mathrm{pr}(D) = \mathrm{monpr}(D)$ for any symmetric digraph D. In this chapter, we prove that the result holds for any digraph.

We also propose a directed graph decomposition, the *Process decomposition*, that is equivalent to the *Process game*. Moreover, the equivalency of the Process game with this decomposition allows us to prove that $\mathrm{pr}(D) = \mathrm{pr}(\overleftarrow{D})$ for any digraph $D = (V, A)$, where $\overleftarrow{D} = (V, \overleftarrow{A})$ and $\overleftarrow{A} = \{(a, b) \mid (b, a) \in A\}$.

3.2 Recontamination Does Not Help to Process a Digraph

In this section, we prove that the process number is monotone. In other words, $\mathrm{monpr}(D) = \mathrm{pr}(D)$ for any directed graph D. For this purpose, we use the techniques introduced in [ST93] and adapted for directed graphs in [Bar06]. More precisely, we first define the notion of a mixed processing game and show its monotonicity thanks to an intermediate result dealing with crusades. Then, from any mixed process strategy we construct a process strategy with the same number of agents in a way that monotonicity is preserved.

Preliminary Definitions

Throughout this section, we use the following notations. Let $D = (V, A)$ be a digraph. For any $v \in V$, let $N^-(v)$ denote the set of in-neighbors of v. The *border* of a set $X \subseteq A$, denoted by $\delta(X)$, is the set of vertices that are the head of an arc in X and the tail of an arc in $A \setminus X$. For any $X \subseteq A$, X^c denotes $A \setminus X$. First, we show that the border function δ is submodular.

Lemma 1. *For any digraph D and any $X, Y \subseteq A(D)$:*

$$|\delta(X \cap Y)| + |\delta(X \cup Y)| \leq |\delta(X)| + |\delta(Y)|.$$

Proof. We show that every vertex counted in the left side of the equation is counted at least the same amount of times in the right side of the equation. Let $v \in \delta(X \cup Y) \cup \delta(X \cap Y)$.

If $v \in \delta(X \cap Y)$, let $e_1 = (u, v) \in X \cap Y$ and $e_2 = (v, w) \in X^c \cup Y^c$. Therefore, either $(v, w) \in X^c$ and $v \in \delta(X)$, or $(v, w) \in Y^c$ and $v \in \delta(Y)$. If $v \in \delta(X \cup Y)$, let $e_1 = (u, v) \in X \cup Y$ and $e_2 = (v, w) \in X^c \cap Y^c$. Therefore, either $(u, v) \in X$ and $v \in \delta(X)$, or $(u, v) \in Y$ and $v \in \delta(Y)$.

Finally, let us assume that $v \in \delta(X \cup Y) \cap \delta(X \cap Y)$. Because $v \in \delta(X \cap Y)$, there exists an edge $e_1 = (u, v) \in X \cap Y$ and because $v \in \delta(X \cup Y)$, there exists an edge $e_2 = (v, w) \in X^c \cap Y^c$. Hence, $v \in \delta(X) \cap \delta(Y)$. □

Let $D = (V, A)$ be a digraph in which no arcs are initially processed. A *mixed process strategy* of D is a sequence $(s_1, \ldots, s_n)$ with the following actions that results in processing all arcs in A.

R_1 **(Place):** place a searcher at an unoccupied node $v \in V$;

R_2 **(Remove):** remove a searcher from node $v \in V$; if there were unprocessed arcs with tail v and v is now unoccupied, then *recontamination* occurs. That is, successively, any processed arc $(u, w) \in A$ becomes unprocessed, if there is w which is unoccupied and an unprocessed arc (w, z).

R_3 **(Head):** process an arc $(u, v) \in A$ if $v \in V$ is *occupied*;

R_4 **(Slide):** slide the searcher at u along $(u, v) \in A$ if u is occupied, v is not occupied and all arcs $e \neq (u, v)$ with tail u are already processed, this processes the arc (u, v);

R_5 **(Extend):** process an arc $(u, v) \in A$, if all arcs with tail v are already processed.

The number of searchers used by a mixed process strategy is the maximum number of occupied vertices over all steps of the strategy. The *mixed process number*, denoted by $\mathrm{mpr}(D)$, is the fewest number of searchers such that there exists a mixed process strategy of D. Moreover, in the mixed *Process game*, we say that a vertex, v, is processed if all edges with tail v are processed. A mixed process strategy is *monotone* if no recontamination occurs, i.e., once an arc has been processed, it must remain processed until the end of the strategy.

Next, we recall the definition of crusades used in [Bar06] and give these crusades an appropriate border function to work with the mixed *Process game*.

A *crusade* in $D = (V, A)$ is a sequence $(X_0, X_1, \ldots, X_n)$ of subsets of A such that $X_0 = \emptyset$, $X_n = E$, and $|X_i \setminus X_{i-1}| \leq 1$, for $1 \leq i \leq n$. The crusade has *border* k if $|\delta(X_i)| \leq k$ for $0 \leq i \leq n$.

A crusade is *progressive* if $X_0 \subset X_1 \subset \ldots \subset X_n$. Hence, in a progressive crusade $(X_0, X_1, \ldots, X_n)$, $|X_i \setminus X_{i-1}| = 1$ for all $i \leq n$.

Intuitively, the elements of a crusade represent the set of edges that are processed, while the border represents the vertices that must be occupied by agents, in order to avoid the processed edges becoming unprocessed.

Note that the notion of mixed strategy and crusade are different from the ones defined in [Bar06], since the direction of the arcs and the border function are reversed in the *Process game*.

Monotonicity

We are ready to show our main result which states that the *Process game* is monotone.

Roughly, we do this by showing that, from any mixed strategy with k searchers, we can obtain a crusade with border k and that, from any crusade with border k, we can obtain a progressive crusade with border k. Then, we show that the existence of a progressive crusade with border k implies the existence of a monotone mixed process strategy with k searchers. Finally, we show how to obtain a monotone process strategy for a directed graph D from a monotone mixed process strategy of $\vec{D}$ (the graph obtained by adding a copy of every arc of D) that uses the same amount of agents. Then, the result follows from the fact that $\mathrm{mpr}(\vec{D}) \leq \mathrm{pr}(D)$.

Lemma 2. *Let D be a digraph. If $\mathrm{mpr}(D) \leq k$, then D admits a crusade with border k.*

Proof. Let $S = (s_1, \ldots, s_n)$ be a mixed process strategy of $D = (V, A)$ that uses at most k searchers. For any $0 < i \leq n$, let A_i be the set of processed arcs and Z_i be the set of occupied vertices after the step s_i. Moreover, let $A_0 = Z_0 = \emptyset$.

By definition of a mixed process strategy, at most one arc is processed in each step s_i (one arc is processed if s_i corresponds to R_3, R_4 or R_5), hence $|A_i \setminus A_{i-1}| \leq 1$ for any $1 \leq i \leq n$. After the last step s_n of S, all the arcs of the graph must be processed, hence $A_n = A$. This proves that $C = (A_0, \ldots, A_n)$ is a crusade.

It remains to show that $\delta(A_i) \leq k$ for every $0 \leq i \leq n$. To do so, we prove by induction that $\delta(A_i) \subseteq Z_i$ for any $1 \leq i \leq n$. It is clearly true for $i = 0$. Assume that $\delta(A_{i-1}) \subseteq Z_{i-1}$ for some i, $0 \leq i < n$. We prove that $\delta(A_i) \subseteq Z_i$:

- If s_i is R_1 (Place), then $A_i = A_{i-1}$ and thus $\delta(A_i) = \delta(A_{i-1}) \subseteq Z_{i-1} \subseteq Z_i$.
- If s_i is R_2 (Remove) at a vertex v, let u be a vertex of $\delta(A_i)$, hence there is an arc $e_1 = (w_1, u) \in A_i$ and an arc $e_2 = (u, w_2) \in A \setminus A_i$, therefore $u \in Z_i$, otherwise e_1 would also become unprocessed in step i making $u \notin \delta(X_i)$, hence $\delta(A_i) \subseteq Z_i$.
- If s_i is R_3 (Head), then $A_i = A_{i-1} \cup \{(u, v)\}$ and $\delta(A_i) \setminus \delta(A_{i-1}) \subseteq \{v\}$. Since v must be occupied, we have $v \in Z_i = Z_{i-1}$, by induction $\delta(A_{i-1}) \subseteq Z_{i-1}$, and therefore $\delta(A_i) \subseteq Z_i$.
- If s_i is R_4 (Slide) at an edge $e = (u, v)$, then $A_i = A_{i-1} \cup \{(u, v)\}$ and $Z_i = (Z_{i-1} \setminus \{u\}) \cup \{v\}$. Since all arcs with tail u are processed after this step, $u \notin \delta(A_i)$. Moreover, $\delta(A_i) \setminus \delta(A_{i-1}) \subseteq \{v\}$. Hence $\delta(A_i) \subseteq Z_i$.
- If s_i is R_5 (Extend), then $A_i = A_{i-1} \cup \{(u, v)\}$ and $Z_i = Z_{i-1}$. Since all arcs with tail v must be already processed, $\delta(A_i) = \delta(A_{i-1}) \subseteq Z_{i-1} = Z_i$.

□

Lemma 3. *If there is a crusade of $D = (V, A)$ with border k, then there is a progressive crusade with border k.*

Proof. Let $C = (X_0, \ldots, X_n)$ be a crusade of D with border k such that: $\sum_{i=0}^{n} |\delta(X_i)|$ is minimum, and subject to this, $\sum_{i=0}^{n} |X_i|$ is minimum. We show that C is progressive. Let $0 < i \leq n$, we show that $X_{i-1} \subset X_i$:

- Assume first that $|X_i \setminus X_{i-1}| = 0$, then $X_i \subseteq X_{i-1}$. Hence, $(X_0, \ldots, X_{i-1}, X_{i+1}, \ldots, X_n)$ is a crusade with border k, contradicting the minimality of $\sum_{i=0}^{n} |X_i|$. Thus, $|X_i \setminus X_{i-1}| = 1$.
- Then assume that $|\delta(X_{i-1} \cup X_i)| < |\delta(X_i)|$, hence $(X_0, \ldots, X_{i-1}, X_{i-1} \cup X_i, X_{i+1}, \ldots, X_n)$ is a crusade with at most k searchers, contradicting the minimality of $\sum_{i=0}^{n} |\delta(X_i)|$. Therefore $|\delta(X_{i-1} \cup X_i)| \geq |\delta(X_i)|$.
- By Lemma 1, $|\delta(X_{i-1} \cap X_i)| + |\delta(X_{i-1} \cup X_i)| \leq |\delta(X_{i-1})| + |\delta(X_i)|$. Hence, by previous item, $|\delta(X_{i-1} \cap X_i)| \leq |\delta(X_{i-1})|$. Therefore, $(X_0, \ldots, X_{i-2}, X_{i-1} \cap X_i, X_i, \ldots, X_n)$ is a crusade with at most k searchers. From the minimality of $\sum_{i=0}^{n} |X_i|$ we have that $|X_{i-1} \cap X_i| \geq |X_{i-1}|$, hence $X_{i-1} \subset X_i$.

□

Lemma 4. *If there is a progressive crusade of $D = (V, A)$ with border k, then there is a* monotone *mixed process strategy using at most k searchers.*

Proof. Let $C = (X_0, \ldots, X_n)$ be a progressive crusade of D using at most k searchers. We build a a monotone mixed process strategy $S = (s_1, \ldots, s_{n'})$ of D with the following properties. For any $0 < i \leq n'$, let A_i be the set of processed arcs and let Z_i be the set of occupied vertices after step s_i. Let $A_0 = Z_0 = \emptyset$. There are $0 = j_0 < j_1 < j_2 < \cdots < j_n = n'$ such that:

1. for any $0 \leq i \leq n$, $A_{j_i} = X_i$;
2. for any $0 < i \leq n$ and for any $j_{i-1} < \ell < j_i$, $Z_\ell \subseteq \delta(X_i)$ or $Z_\ell \subseteq \delta(X_{i-1})$, and $Z_{j_i} = \delta(X_i)$.

Starting with $S = \emptyset$, the two above properties hold for $i = 0$. Let $0 < i \leq n$ and let us assume that $(s_1, \ldots, s_{j_{i-1}})$ is a sequence of actions that satisfies the two above properties for any $0 \leq j < i$. We will build the next steps of the strategy until s_{j_i}. Let $X_i \setminus X_{i-1} = \{e_i\}$, where $e_i = (u, v)$. Note that $\delta(X_i) \setminus \delta(X_{i-1}) \subseteq \{v\}$ and $\delta(X_{i-1}) \setminus \delta(X_i) \subseteq \{u, v\}$. We have several cases to consider:

- let us first assume that $v \in \delta(X_{i-1})$. Hence, $v \in Z_{j_{i-1}}$ and there is a searcher at v after step $s_{j_{i-1}}$. We define the step $s_{j_{i-1}+1}$ to be R_3 (Head) at e_i, i.e., the arc e_i is processed.
  - If moreover $v \notin \delta(X_i)$ then we define the step $s_{j_{i-1}+2}$ to be R_2 at v, i.e., we remove the searcher at v. Because $v \notin \delta(X_i)$ and (u, v) is processed, there are no unprocessed arcs with tail v and therefore, no recontamination occurs.

  Let $k = j_{i-1} + 3$ if $v \notin \delta(X_i)$ and $k = j_{i-1} + 2$ otherwise.

– Finally, if $u \in \delta(X_{i-1}) \setminus \delta(X_i)$, then we define the step s_k to be R_2 at u, i.e., we remove the searcher at u. Because $u \in \delta(X_{i-1})$, there is an arc with head u that was processed after step $s_{j_{i-1}}$. Because $u \notin \delta(X_i)$ and (u, v) is processed, there are now no unprocessed arcs with tail u and therefore, no recontamination occurs.

Hence, $j_{i-1}+1 \leq j_i \leq j_{i-1}+3$. Clearly, for any $j_{i-1}+1 \leq \ell \leq j_{i-1}+3$, $Z_\ell \subseteq \delta(X_{i-1})$ and $\delta(X_i) = Z_{j_i}$ in all cases. Moreover, in all cases, no recontamination occurs. Therefore, $A_{j_i} = X_i$.

- Now, let us assume that $v \notin \delta(X_{i-1})$. By induction, there was no searcher at v after step $s_{j_{i-1}}$.

  – First, let us consider the case when $u \in \delta(X_{i-1})$:

    * if $v \in \delta(X_i)$ and $u \in \delta(X_i)$: Let us define the step $s_{j_{i-1}+1}$ to be R_1 at v, i.e., a searcher is placed at v, and the step $s_{j_i} = s_{j_{i-1}+2}$ is defined as R_3 (Head) at e_i, i.e., the edge e_i is processed. Clearly, $A_{j_i} = A_{j_{i-1}} \cup \{e_i\}$ and $Z_{j_{i-1}+1} = Z_{j_i} = Z_{j_{i-1}} \cup \{v\} = \delta(X_i)$.
    * if $v \in \delta(X_i)$ and $u \notin \delta(X_i)$: in that case, the only arc in $A \setminus X_{i-1}$ which has u as tail is e_i, otherwise $u \in \delta(X_i)$. Therefore we define the step $s_{j_{i-1}+1} = s_{j_i}$ to be R_4 through e_i, i.e., the searcher at u slides to v processing e_i. Note that no recontamination occurs and $A_{j_i} = A_{j_{i-1}} \cup \{e_i\} = X_{i-1} \cup \{e_i\} = X_i$. The induction hypothesis holds since $Z_{j_i} = (Z_{j_{i-1}} \setminus \{u\}) \cup \{v\} = (\delta(X_{i-1}) \setminus \{u\}) \cup \{v\} = \delta(X_i)$.
    * if $v \notin \delta(X_i)$ then there are no arcs with tail v that are in $A \setminus X_{i-1}$. Hence, we can define the step $s_{j_{i-1}+1}$ to be R_5 (extend) at e_i, i.e., e_i is processed. If moreover $u \notin \delta(X_i)$, let $s_{j_i} = s_{j_{i-1}+2}$ be defined as R_2 at u, i.e., the searcher at u is removed. Because $u \in \delta(X_{i-1}) \setminus \delta(X_i)$ and (u, v) is now processed, there are no unprocessed arcs with tail u and therefore, no recontamination occurs.
      Hence, $j_{i-1} + 1 \leq j_i \leq j_{i-1} + 2$ and the induction hypothesis holds in both cases.

  – Finally, consider the case when $u \notin \delta(X_{i-1})$. Note that, in that case, since $u \notin \delta(X_{i-1})$ and u is a tail of $e_i \in X_i$, then $u \notin \delta(X_i)$.

    * if $v \in \delta(X_i)$ then we define the step $s_{j_{i-1}+1}$ to be R_1 at v, i.e., a searcher is placed at v, and $s_{j_i} = s_{j_{i-1}+2}$ to be R_3 (Head) at e_i, i.e., e_i is processed. The induction hypothesis holds.
    * if $v \notin \delta(X_i)$, since $e_i \in X_i$, then there are no arcs with tail v that are in $A \setminus X_{i-1}$. Hence we can define the step $s_{j_i} = s_{j_{i-1}+1}$ as R_5 (Extend) at e_i, i.e., e_i is processed. The induction hypothesis holds.

Therefore, $S = (s_1, \ldots, s_{j_n})$ satisfies the two properties, and S is a monotone mixed process strategy using at most k searchers in D, since, for all $1 \leq i \leq j_n$, we have that $|Z_i| \leq k$, for all $1 \leq i < j_n$, $A_i \subseteq A_{i+1}$, and $A_{j_n} = X_n = A$. □

In what follows, let $\vec{D}$ be the digraph obtained from any digraph $D = (V, A)$ by adding a copy of every arc of D.

Theorem 5. *For any digraph $D = (V, A)$:*

$$\text{monpr}(D) \leq \text{mpr}(\vec{D}) \leq \text{pr}(D).$$

Proof. We first show that $\text{mpr}(\vec{D}) \leq \text{pr}(D)$.

Let $S^p = (s_1, \ldots, s_n)$ be a process strategy for D using k searchers. We define a mixed process strategy $S^m = (m_1, \ldots, m_j)$ using at most k searchers for $\vec{D}$. Let P_i be the set of processed vertices at step $i \leq n$ in S^p and let M_j be the set of vertices u such that, at step m_j in S^m, all arcs with u as tail are processed. Also, let O_i^p (resp., O_i^m) be the set of vertices occupied by a searcher at step s_i in S^p (resp., at step m_i in S^m).

For any $0 < i \leq n$, we build a phase of S^m according to s_i. That is, depending on the type of rule applied in s_i, we add a sequence of moves $m_{j_{i-1}+1}, m_{j_{i-1}+1}, \ldots, m_{j_i}$ in S^m such that $P_i \subseteq M_{j_i}$. Hence, at the last step all arcs are processed, since $P_n = V$. To do this, assume that $m_1, \ldots, m_{j_{i-1}}$ are already defined based on $(s_1, \ldots, s_{i-1})$ and that $P_{i-1} \subseteq M_{j_{i-1}}$. Moreover, assume that $O_{i-1}^p = O_{j_{i-1}}^m$. We define $m_{j_{i-1}+1}, m_{j_{i-1}+1}, \ldots, m_{j_i}$ depending on which rule is applied in s_i:

- If s_i is a place operation at vertex v (move M_1), then let us define the step $m_{j_{i-1}+1}$ to be R_1 at vertex v, i.e., a searcher is placed at v. Then, let $\{e_1, \ldots, e_r\}$ be the set of arcs with head v. For any $\ell \in [2, r+1]$, let us define the step $m_{j_{i-1}+\ell}$ to be R_3 (Head) at e_ℓ. That is, all arcs with head v are sequentially processed.

  Hence, $j_i = j_{i-1} + r + 1$. The claim holds since $P_i = P_{i-1} \subseteq M_{j_{i-1}} \subseteq M_{j_i}$, and moreover, for any $j_{i-1} < \ell \leq j_i$, $O_\ell^m = O_i^p = O_{i-1}^p \cup \{v\}$.

- If s_i consists in processing an unoccupied vertex v (move M_2), then after step s_{i-1} in S^p, all vertices that are in the out-neighborhood of v are already processed. Hence, by the construction of S^m, after step $s_{j_{i-1}}$ in S^m, all arcs with tail v are already processed. Moreover, because $v \notin O_{i-1}^p = O_{j_{i-1}}^m$ then v is also unoccupied at step j_{i-1} of S^m.

  Hence, let $\{e_1, \ldots, e_r\}$ be the set of arcs with head v. For any $1 \leq \ell \leq r$, let us define $m_{j_{i-1}+\ell}$ as R_5 (Extend) at e_i. That is, all arcs with head v are sequentially processed.

  In that case, $j_i = j_{i-1} + r$. The claim holds, since, in particular, $v \in M_{j_{i-1}}$.

- Now consider the case when s_i consists in processing an occupied vertex v and removing the searcher at v (move M_3'). Let us define the step $m_{j_{i-1}+1} = m_{j_i}$ to be R_2 at v, i.e., the searcher at v is removed. In the case of recontamination in S^m, all vertices, v, in $v \in M_{j_{i-1}} \setminus M_{j_i}$ are tail of some arc e such that there is a path from v avoiding agents and passing through e that reaches an unprocessed arc in $\vec{D}$. Therefore, v also becomes unprocessed in S^p, i.e. $v \in P_{i-1} \setminus P_i$. Hence, the claim holds.

Therefore, S^m is a mixed process strategy for D using at most k searchers.

Now, let us show that $\text{monpr}(D) \leq \text{mpr}(\vec{D})$. By Lemmas 2, 3 and 4, there exists a monotone mixed process strategy using $\text{mpr}(\vec{D})$ searchers in $\vec{D}$. Let $S^m = (m_1, \ldots, m_n)$ be such a strategy.

We first notice that if there is a step m_i $(1 \leq i \leq n)$ that applies a rule of type R_4 (Slide) through an arc $e_1 = (u, v)$, then the second arc $e_2 = (u, v)$ must be processed and

there must be no searcher at v. Hence it is possible to replace the step m_i by the following: first remove the agent from u, without re-contaminating any arc (since otherwise e_2 would have been re-contaminated before), place the agent at v and apply R_3 (Head) operation at e_1. Therefore, we may assume that S^m never applies moves of type R_4.

Another remark is that, if the step m_i consists in processing an arc (u, v) such that u is occupied and all arcs with u as tail are already processed, then we may assume that the step m_{i+1} applies the rule R_2 to u, i.e., the searcher at u is removed (and no recontamination occurs). Indeed, after step m_i, the searcher at u is not used to preserve from recontamination because the strategy is monotone and all its outgoing arcs are processed. Moreover, if this searcher was used to process one in-coming arc of u at a step further, we can instead use the extend rule R_5. Finally, by previous remark, this searcher is never used to apply rule R_4.

Let M_i be the set of *unoccupied* vertices u such that all arcs with tail u are already processed after step m_i.

We now define a monotone process strategy $S^p = (s_1, \ldots, s_n)$ for D that uses at most $\mathrm{mpr}(\vec{D})$ searchers. Let P_i be the set of processed vertices at step $i \leq n$ in S^p and let M_i be the set of *unoccupied* vertices u such that all arcs with tail u are already processed after step m_i in S^m. Also, let O_i^p, resp., O_i^m, be the set of vertices occupied by a searcher at step s_i in S^p, resp., at step m_i in S^m. Assume that $(s_1, \ldots, s_{j_{i-1}})$ is already defined such that $O_{i-1}^m = O_{i-1}^p$, and $M_{i-1} \subseteq P_{i-1}$ or ($M_{i-1} \subseteq P_{i-1} \cup \{v\}$ and m_i consists in removing a searcher from some node v). We define s_i depending on m_i:

- Assume first that m_i consists in placing a searcher at vertex v (R_1). Then, let s_i consist in placing a searcher at v (M_1). The claim holds, since $M_i \subseteq M_i$ and $O_i^p = O_{i-1}^p \cup \{v\} = O_{i-1}^m \cup \{v\} = O_i^m$.

- If m_i consists in removing a searcher from a vertex v (R_2) then, since S^m is monotone, recontamination does not happen. That is, there are no unoccupied directed path from a process arc to an unprocessed one. Note that v is occupied since $O_{i-1}^m = O_{i-1}^p$. In that case, let s_i consists in processing v and removing the searcher at v (M_3), this is possible since all out-neighbors of v are either occupied or processed in S^m.

  The claim holds since $P_{j_i} = P_{j_{i-1}} \cup \{v\}$ and $M_i = M_{i-1} \cup \{v\}$, and moreover, $O_i^p = O_{i-1}^p \setminus \{v\} = O_{i-1}^m \setminus \{v\} = O_i^m$.

- If m_i consists in processing an arc $e = (u, v) \in A(D)$ (R_3 or R_5). Then, if e is the only unprocessed arc with tail u before m_i:

  - If u is occupied by the remark above, the next step m_{i+1} consists in removing the searcher at u. In that case, s_i consists in doing nothing and we have $M_i \subseteq P_i \cup \{u\}$ and $O_i^m = O_i^p$.
  - Else, let s_i consists in processing u (applying M_2). Again, the properties hold.

  If m_i consists in processing an arc $(u, v) \in A(D)$ that is not the last unprocessed outgoing arc of u (in particular, we may assume it is the case for all arcs in $A(\vec{D}) \setminus A(D)$), then s_i consists in doing nothing and the properties hold.

Therefore, S^p is a monotone process strategy for D using at most $\mathrm{mpr}(\vec{D})$ searchers. □

Since, for any digraph D, $\mathrm{pr}(D) \leq \mathrm{monpr}(D)$, we obtain the next corollary:

Corollary 1. *recontamination does not help to process a digraph, i.e., for any digraph D:*

$$\mathrm{pr}(D) = \mathrm{monpr}(D).$$

Corollary 1 shows that the *Process game* is monotone. That is, allowing vertices to become unprocessed does not help the agents to process the graph.

3.3 Process Decomposition

In this section we define a digraph decomposition that is equivalent to (monotone) process strategies. This allows us to prove that the process number is invariant when reversing all arcs of a digraph. Let $D = (V, A)$ be a digraph.

A *Process Decomposition* of D is a sequence of pairs $P = ((W_1, X_1), \ldots, (W_t, X_t))$ such that:

- for any $1 \leq i \leq t$, $W_i \subseteq V$ and $X_i \subseteq V$;
- $(X_1, \ldots, X_t)$ is a partition of $V \setminus \bigcup_{i=1}^{t} W_i$;
- $\forall i \leq j \leq k$, $W_i \cap W_k \subseteq W_j$;
- X_i induces a Directed Acyclic Graph (DAG), for any $1 \leq i \leq t$;
- $\forall (u, v) \in A$, $\exists j \leq i$ such that $v \in W_j \cup X_j$ and $u \in W_i \cup X_i$.

The width of a *Process Decomposition* is given by $\max_{1 \leq i \leq n} |W_i|$, and the *process width*, denoted by $\mathrm{prw}(D)$, of a digraph D is given by the minimum width over all *Process Decompositions* of D. A Scheme of a *Process Decomposition* can be found in Figure 3.1.

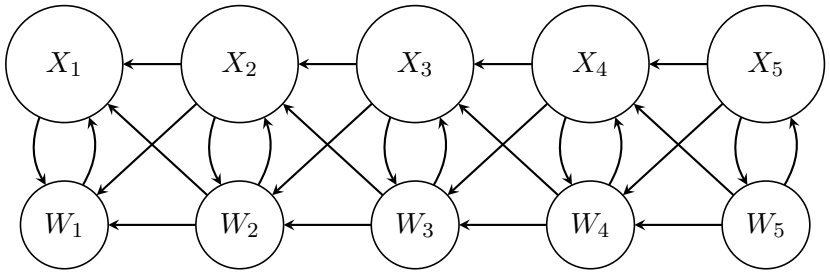


Figure 3.1: A scheme of the *Process Decomposition*. Sets X_i are disjoint with each inducing a DAG in the original graph. The sets W_i behave in a similar manner as the bags in a directed path decomposition.

A first result shows that reversing the arcs of a digraph does not change its process width. Let $\overleftarrow{D}$ be the digraph obtained by reversing the sense of the arcs of a digraph $D = (V, A)$.

Lemma 6. *For any digraph D:*

$$\mathrm{prw}(D) = \mathrm{prw}(\overleftarrow{D}).$$

Proof. Let $P = ((W_1, X_1), \ldots, (W_t, X_t))$ be a *Process Decomposition* for D with width w. Let $\overleftarrow{P} = ((W_t, X_t), \ldots, (W_1, X_1))$. Clearly, the first four properties of *Process Decomposition* hold, and the width of $\overleftarrow{P}$ is w.

It remains to show that $\forall (u, v) \in A(\overleftarrow{D})$, $\exists i \leq j$ such that $u \in W_i \cup X_i$ and $v \in W_j \cup X_j$. To do that, consider an edge $(u, v) \in A(\overleftarrow{D})$, since P is a *Process Decomposition* of D and $(v, u) \in A(D)$, we have that for some $j \leq i$, $v \in W_j \cup X_j$ and $u \in W_i \cup X_i$, therefore $\overleftarrow{P}$ is a *Process Decomposition* of $\overleftarrow{D}$. □

Theorem 7. *For any digraph D:*

$$\mathrm{pr}(D) = \mathrm{prw}(D).$$

Proof. We have that $\mathrm{pr}(D) = \mathrm{monpr}(D)$, by Theorem 1. Hence, we only need to show that $\mathrm{monpr}(D) = \mathrm{prw}(D)$.

To show that $\mathrm{prw}(D) \geq \mathrm{monpr}(D)$ let $P = ((W_1, X_1), \ldots, (W_t, X_t))$ be a *Process Decomposition* of D with width w. We construct a monotone process strategy of D using at most w searchers. For any $1 \leq i \leq t$, we define the sequence of moves (Phase i) from (W_i, X_i), such that, after this sequence, the vertices of $W_i \cap W_{i+1}$ are occupied by searchers and the vertices in $\bigcup_{j=1}^{i}(W_j \cup X_j) \setminus W_{i+1}$ have been processed.

At phase $i+1$, we first place searchers at the vertices of $W_{i+1} \setminus W_i$. Then, in the inverse of a topological ordering of the DAG induced by X_{i+1}, vertices of X_{i+1} are processed. This is possible because, for any vertex v in X_{i+1}, any out-neighbor u of v is in $\bigcup_{j=1}^{i+1} X_j \cup W_j$ and so u is either already processed or occupied. Finally, searchers are removed from the vertices in $W_{i+1} \setminus W_{i+2}$ and these vertices are processed. Again, this is valid since all out-neighbor of a vertex in $W_{i+1} \setminus W_{i+2}$ belongs to $\bigcup_{j=1}^{i+1} X_j \cup W_j$ (by the last property of the decomposition).

Clearly, such a strategy is monotone and uses at most w searchers, hence $\mathrm{monpr}(D) \leq \mathrm{prw}(D)$.

Now, let us show that $\mathrm{monpr}(D) \geq \mathrm{prw}(D)$. Let $S = (s_1, \ldots, s_t)$ be a monotone process strategy of D using k searchers. We remark that if a searcher is removed from a vertex v, this vertex is also processed during the same step. We construct a *Process Decomposition* of D with width at most k. For any $1 \leq i \leq t$, let (W_i, X_i) be defined as follows. Let $(W_0, X_0) = (\emptyset, \emptyset)$:

M_1**:** if s_i consists in placing a searcher at vertex v, then $W_i = W_{i-1} \cup \{v\}$ and $X_i = \emptyset$;

M_2**:** if s_i consists in processing an unoccupied vertex v, then $W_i = W_{i-1}$ and $X_i = \{v\}$;

M_3**:** if s_i consists in processing an occupied vertex v and removing the searcher at v, then $W_i = W_{i-1} \setminus \{v\}$ and $X_i = \emptyset$.

It is easy to see that $(X_1, \ldots, X_t)$ is a partition of $V \setminus \bigcup_{i=1}^{t} W_i$ since all vertices are either occupied or processed (only once) without being occupied. Moreover, any X_i being reduced to at most a singleton induces a DAG. By the rules of the monotone process strategy, any vertex is occupied at most once (i.e., there are no two steps of S that consist in placing a searcher at the same vertex), and so $\forall i \leq j \leq k$, $W_i \cap W_k \subseteq W_j$.

Finally, let $(u, v) \in A$ and let i be the greatest integer such that $u \in W_i \cup X_i$ and let j be the smallest integer such that $v \in W_j \cup X_j$. For purpose of contradiction, assume that $j > i$. Then, u is processed at step s_i while its out-neighbor v is neither processed nor occupied at step i, since $j > i$, a contradiction.

Clearly, $\max_{i \leq t} |W_i| \leq k$. □

Remark 1. By the proof of Theorem 7, for any digraph D, there is an optimal *Process Decomposition* $((W_1, X_1), \ldots, (W_t, X_t))$ of D such that X_i has size at most one for any $i \leq t$.

Corollary 2. *Given a digraph $D = (V, A)$ and $\overleftarrow{D}$, the graph obtained from D by reversing all the arcs, then:*

$$\text{monpr}(D) = \text{monpr}(\overleftarrow{D}) = \text{pr}(D) = \text{pr}(\overleftarrow{D}).$$

3.4 Conclusion

One of the main results in this chapter is the monotonicity of the *Process game*. One consequence of this result is that, in the routing reconfiguration problem, allowing connections also to be re-established back into their initial routing does not help reroute the network. That is, the minimum number of maximum simultaneously interrupted connections does not change by allowing this rule.

We also propose a directed graph decomposition, the *Process Decomposition*, that is equivalent to the *Process game*. Since finding good strategies for the searchers in the *Process game* is NP-hard in general, another focus could be to study the behavior of the *Process game* into specific classes of directed graphs. Because the *Process Decomposition* gives us a global vision on how a strategy can process a graph, we expect it to be a major building block in the construction of strategies for the *Process game* in specific classes of graphs.

Another problem that arises with the introduction of the *Process Decomposition* is how to compute such decomposition of a graph. Is there an FPT-algorithm for computing a *Process Decomposition* of a digraph? Albeit this question will be left unanswered in this thesis, in the next chapter, we explore the problem of computing graph decompositions for undirected graphs.

Finally, both tree decompositions and path decompositions have the notion of a dual structure, brambles and blockages respectively. For instance, the tree width of a undirected graph G is equal to $k-1$ if, and only if, G has no bramble greater[3] than k [ST93]. The monotonicity of a game plays an important role in the relationship between the width of a decomposition and its dual. Hence, it will be interesting to use our monotonicity result to define a dual for the process number.

In addition, the visible variant of the processing game appears to be an interesting candidate for providing a "tree like" decomposition for digraphs.

[3]Measured by the size of its hitting set.

Chapter 4

Graph Width Measures

In this chapter, we aim at investigating the problem of computing undirected graph decompositions.

More precisely, this chapter aims at unifying and generalizing the FPT algorithms for computing various decompositions of graphs. As a particular application, our algorithm decides in linear time if the special tree width or the q-branched tree width is at most k, for $q \geq 0$ and $k \geq 1$ fixed and, hence, being the first explicit FPT-algorithm capable of computing these parameters.

We use the notions of partition function and partitioning-tree, defined in [AMNT09], in order to generalize these algorithm. Given a finite set A, a partition function Φ for A is a function from the set of partitions of A into the integers. A partitioning-tree of A is a tree T together with a one-to-one mapping between A and the leaves of T. Note that, every internal vertex v of T defines a partition of A where each part is composed by the leaves of T in one component of $T[V(T) \setminus \{v\}]$. The Φ-width of T is the maximum $\Phi(\mathcal{P})$, for any partition $\mathcal{P}$ of A defined by the internal vertices of T, and the Φ-width of A is the minimum Φ-width of its partitioning-trees. Partition functions are a unified view for a large class of width parameters like tree width, path width, branch width, etc. In [AMNT09] is given a simple sufficient property that a partition function for A must satisfy to ensure that either A admits a partitioning tree of width at most $k \geq 1$, or there exists a *k-bramble* (a dual structure), unifying and generalizing the duality theorems in [RS83, RS91, ST93, FT03].

We propose a simple set of sufficient properties and an algorithm such that, for any k and q fixed parameters, and any partition function Φ satisfying these properties, our algorithm decides in time $O(|A|)$ if a finite set A has q-branched Φ-width at most k (Theorem 11). Since tree width, path width, branch width, cut width, linear width, and carving width can be defined in terms of Φ-width for some particular partition functions Φ that satisfy our properties (Theorem 12), our algorithm unifies the works in [Bod96, BT97, Thi00, BT04]. Our algorithm generalizes the previous algorithms since it is not restricted to width-parameters of graphs, but works as well for any partition function (not restricted to graphs) satisfying some simple properties. Moreover, we show how the special tree width [Cou10] of a graph can be defined by a partition function that satisfy our properties. This implies that our algorithm can also be used to decide, in linear time, if the special tree width of a graph is not bigger than a constant k. Finally, it provides the first explicit linear-time algorithm that decides if a graph G can be searched,

in a non-deterministic way, by k searchers performing at most q queries, for any $k \geq 1$, $q \geq 0$ fixed. Our decision algorithm can be turned into a constructive one by following the ideas of Bodlaender and Kloks [Bod96].

In Section 4.1, we formally define the notions of partition functions and partitioning-trees. Then, we present several width parameters of graphs in terms of partition functions (most of these results have been proved in [AMNT09]). Section 4.2 is devoted to the formal statement of our results. In Section 4.3, we show a method to describe all partitioning trees of Φ-width not bigger than k of a set A. Section 4.4 is dedicated to show how partitioning trees can be represented in an efficient manner. Then, in Section 4.5, we describe an algorithm that follows the method in Section 4.3, but using the efficient representation of partitioning trees of Section 4.4, to decide if a set A has Φ-width at most k, Φ being a partition function and k a fixed integer. Then, in Section 4.6, we briefly discuss the results in this chapter with some perspectives into future work.

4.1 Partition Functions and Partitioning Trees

In this section, we present the notions of partition function and partitioning tree of a set, as defined in [AMNT09].

Let A be a finite set. A partition of A is a set of non-empty pairwise disjoint subsets of A whose union equals A. Let $\mathrm{Part}(A)$ be the set of all partitions of A. Let $\mathcal{P} = (A_i)_{i \leq r}$ and $\mathcal{Q} = (B_i)_{i \leq p}$ be two partitions of A. For any subset $A' \subseteq A$, the *restriction* $\mathcal{P} \cap A'$ of $\mathcal{P}$ to A' is the partition $(A_i \cap A')_{i \leq r}$ of A', with its empty parts removed. $\mathcal{Q}$ is a *subdivision* of $\mathcal{P}$ if, for any $j \leq p$, there exists $i \leq r$ with $B_j \subseteq A_i$.

Definition 1. A *partition function* Φ_A *for* A is a function from $\mathrm{Part}(A)$ into the integers. A partitioning function Φ_A is *monotone* if, for any subdivision $\mathcal{Q}$ of a partition $\mathcal{P}$ of A, $\Phi_A(\mathcal{P}) \leq \Phi_A(\mathcal{Q})$.

For the purpose of generalization, we would like a partition function to be defined independently from the set on which it is applied. In particular, we would like that a partition function, for some set A, to induce some partition functions for any subset of A.

From now on, $\mathcal{A}$ denotes a set of finite sets closed under taking subsets. In other words, $\forall X \in \mathcal{A}$ and $\forall Y \subseteq X$ we have $Y \in \mathcal{A}$.

Definition 2. A (monotone) *partition function* Φ *over* $\mathcal{A}$ is a function that associates a (monotone) partition function Φ_A for A to any $A \in \mathcal{A}$.

Most of the results of the chapter do not depend on the set A. When this is the case, we simplify the notation of a *partition function* Φ by omitting the subscript.

Definition 3. A partition function Φ over $\mathcal{A}$ is *closed under taking subsets* if Φ associates a partitioning function $\Phi_{A'}$ for any $A' \subseteq A \in \mathcal{A}$ and, for any partition $\mathcal{P}$ of A, $\Phi_{A'}(\mathcal{P} \cap A') \leq \Phi_A(\mathcal{P})$.

In what follows, we define partitioning trees.

Definition 4. A *partitioning tree* (T, σ) of a set A is a tree T together with a one-to-one mapping σ between the elements of A and the leaves of T.

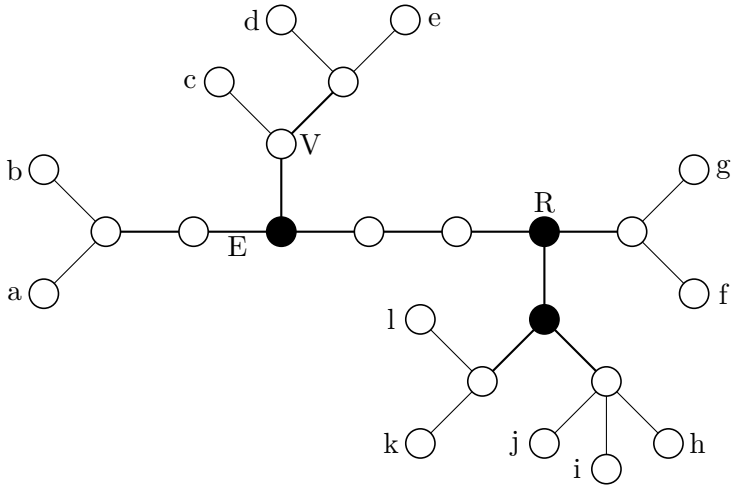
If T is rooted in $r \in V(T)$, the partitioning tree is denoted by (T, r, σ). Any internal (i.e. non leaf) vertex $v \in V(T)$ corresponds to a partition $\mathcal{T}_v$ of A, defined by the sets of leaves of the connected components of $T \backslash v$. Figure 4.1 shows an example of a partitioning tree. Similarly, any edge $e \in E(T)$ defines a bi-partition $\mathcal{T}_e$ of A. The Φ_A *width* of (T, σ) is the maximum $\Phi_A(\mathcal{T})$ where $\mathcal{T}$ is the partition defined by an internal vertex of T or an edge of T.

Definition 5. Let Φ be a partition function over $\mathcal{A}$. The Φ-*width* of a set $A \in \mathcal{A}$ is the minimum Φ_A width of its partitioning trees.

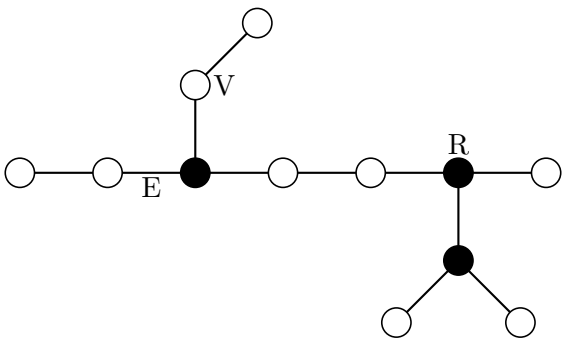
A *branching node* of tree T rooted in $r \in V(T)$ is any vertex of T with at least two children. A tree T is q-branched if there exists a root $r \in V(T)$ such that any path from r to a leaf contains at most $q \geq 0$ branching nodes. For instance, T is 0-branched if and only if T is a path.

Definition 6. The *corpse* $\mathrm{cp}(T)$ of a tree T rooted in $r \in V(T)$ denotes the tree rooted in r obtained from T by removing all its leaves, but r if it is a leaf.

In Figure 4.1, a 2-branched partitioning tree (T, R, σ) of the elements $a, b, \ldots, k, l$ is represented. The vertex $V \in V(T)$ defines the partition $\mathcal{T}_V$ with parts $\{abfghijkl, c, de\}$, $R \in V(T)$ defines the partition $\mathcal{T}_R = \{abcde, fg, hijkl\}$, and the edge $E \in E(T)$ defines the bi-partition $\mathcal{T}_E = \{ab, cdefghijkl\}$. The black vertices are the branching nodes of $\mathrm{cp}(T)$.



(a) tree



(b) corpse

Figure 4.1: A partitioning tree of $\{a, b, \ldots, k, l\}$ (a) and its corpse (b).

Definition 7. A partitioning tree (T, σ) is *q-branched* if the corpse $\text{cp}(T)$ of T is q-branched.

For instance, a partitioning tree (T, σ) is 0-branched if and only if T is a caterpillar[1]. The *q-branched* Φ*-width* of A is the minimum Φ_A width of its q-branched partitioning trees.

Graph Decompositions and Partitioning Trees

The notions from Section 4.1 have been given for general sets. In the following, we recall that partition functions and partitioning trees are generalization of several decompositions of graphs and their related parameters [AMNT09]. We assume that the reader is familiar with the width measures of graphs of Chapter 2.

Throughout this section, $\mathcal{E}$ contains all possible edge-sets of every graph, i.e. for any graph $G = (V, E)$ we have $E \in \mathcal{E}$ and $\mathcal{V}$ contains all possible vertex-set of every graph, i.e. for any graph $G = (V, E)$ we have $V \in \mathcal{V}$.

It is sometimes necessary, depending on the width measure, to restrict the shape of the partitioning tree, to add some constraint to the mapping of the leaves of the partitioning tree or to use a special partitioning function in order to express graph width measures in terms of partitioning functions and partitioning trees. In what follows, we show which restrictions are necessary to represent the special tree width, branch width, linear width, cut width and carving width in terms of partitioning functions and partitioning trees. We start by reproducing how the q-branched tree width of a graph can be represented by partitioning functions as shown in [AMNT09].

Partition function and q-branched tree width

For any graph $G = (V, E)$, let Δ be the function that assigns, to any partition $\mathcal{X} = \{E_1, \ldots, E_r\}$ of E, the set of the vertices that are incident to edges in E_i and to edges in E_j, with $i \neq j$.

Definition 8. Let $E \in \mathcal{E}$. The function δ_E is the partition function for E that assigns $|\Delta(\mathcal{X})|$ to any partition $\mathcal{X}$ of $\text{Part}(E)$. Let δ be the partition function over $\mathcal{E}$ that assigns δ_E to every $E \in \mathcal{E}$.

Lemma 8. *[AMNT09] For any graph $G = (V, E)$, the tree width $\text{tw}(G)$ of G is at most $k \geq 1$ if, and only if, the δ-width of E is at most $k + 1$.*

Proof. In other words, we aim at proving that for any graph $G = (V, E)$, the tree width $\text{tw}(G)$ of G is at most k if, and only if, there is a partitioning tree of E with δ width at most $k + 1$. Let (T, σ) be a partitioning tree of E with δ width at most $k + 1$, then it is easy to check that $(\text{cp}(T), (X_t)_{t \in V(\text{cp}(T))})$, with $X_t = \Delta(\mathcal{T}_t)$, is a tree decomposition of G of width at most k. Conversely, let $(T, \mathcal{X})$ be a tree decomposition of G with width at most k. Then, for any edge $\{x, y\} \in E$, let us choose an arbitrary bag X_t that contains both x and y, add a leaf f adjacent to t in T, and let $\sigma(f) = \{x, y\}$. Finally, let S be the minimal subtree spanning all such leaves. The resulting tree (S, σ) is a partitioning tree of E with δ width at most $k + 1$. □

A similar proof leads to:

[1] A caterpillar is a tree with a dominating path.

Lemma 9. *[AMNT09] For any graph $G = (V, E)$, the path width $\mathrm{pw}(G)$ of G is at most $k \geq 1$ if, and only if, there is a 0-branched partitioning tree (T, σ) of E with δ width at most k.*

More generally:

Lemma 10. *For any graph $G = (V, E)$ and any $q \geq 0$, the q-branched tree width $\mathrm{tw}_q(G)$ of G is at most $k \geq 1$ if, and only if, there is a q-branched partitioning tree (T, σ) of E with δ width at most k.*

The special tree width can be represented with the following restriction over the partitioning trees.

Special tree width: The special tree width of a graph can be expressed in terms of the partition function δ, but with a restriction in the shape of the partitioning tree. For any graph $G = (V, E)$, instead of searching for the minimum δ_E width over all partitioning trees of E, we restrict the partitioning trees to respect the following rule. Let (T, σ) be a partitioning tree of E, for each vertex v in V, let T' be the minimum subtree of (T, σ) spanning all the leaves of (T, σ) such that their corresponding edge in E has v as one extremity. We have that T' is a caterpillar.

Other Widths and Partition Functions

The branch width and the linear width of a graph may be expressed in terms of the following partition function:

Definition 9. Let $max\delta_E$ be the partition function for $E \in \mathcal{E}$ which assigns $\max_{i \leq n} \delta(E_i, E \setminus E_i)$ to any partition $(E_1, \ldots, E_n)$ of E. Let $max\delta$ be the partition function over $\mathcal{E}$ that assigns $max\delta_E$ to any $E \in \mathcal{E}$.

Branch width [BT97]: By definition, the branch width of G, denoted by $\mathrm{bw}(G)$, is at most $k \geq 1$, if and only if there is a partitioning tree (T, σ) of E with $max\delta$ width at most k and such that the internal vertices of T have maximum degree at most three.

Linear width [BT04]: The linear width of G, denoted by $\mathrm{lw}(G)$ is defined as the smallest integer k such that E can be arranged in a linear ordering $(e_1, \ldots, e_m)$ such that for every $i = 1, \ldots, m-1$ there are at most k vertices both incident to an edge that belongs to $\{e_1, \ldots, e_i\}$ and to an edge in $\{e_{i+1}, \ldots, e_m\}$. The linear width of G is at most $k \geq 2$ if and only if there is a partitioning tree (T, σ) of E with $max\delta$ width at most k, such that the internal vertices of T have maximum degree at most three, and (T, σ) is 0-branched. This result easily follows from the trivial correspondence between such a partitioning tree of E and an ordering of E, see Chapter 2. Note that this does not hold for $k = 1$ (a 3 edges path is a counterexample).

The carving width of a graph may be expressed in terms of the following partition function. For any partition $\mathcal{X} = \{V_1, \ldots, V_r\}$ of $V \in \mathcal{V}$, let $Edge\delta$ be the function that assigns the cardinality of the set of the edges of the graph $G = (V, E)$ that are incident to vertices in V_i and V_j, with $i \neq j$.

Definition 10. Let $maxEdge\delta_V$ be the partition function for $V \in \mathcal{V}$ that assigns $\max_{i \leq n} Edge\delta(V_i, V \setminus V_i)$ to any partition $(V_1, \ldots, V_n)$ of V. Let $maxEdge\delta$ be the partition function over $\mathcal{V}$ that assigns $maxEdge\delta_V$ to any $V \in \mathcal{V}$.

Carving width [ST94, Thi00]: By definition, the carving width of G, carw(G), is at most $k \geq 1$ if and only if there is a partitioning tree of V with *maxEdgeδ* width at most k, and such that the internal vertices of T have maximum degree at most three.

The cut width of G, denoted by cw(G), is defined as the smallest integer k such that V can be arranged in a linear ordering $(v_1, \ldots, v_n)$ such that for every $i = 1, \ldots, n-1$ there are at most k edges both incident to a vertex that belongs to $\{v_1, \ldots, v_i\}$ and to a vertex in $\{v_{i+1}, \ldots, v_n\}$.

The partition function *maxEdgeδ* also may express the cut width of a graph $G = (V, E)$ when it is at least the maximum degree Δ of G. Note that any 0-branched partitioning tree with maximum degree at most 3 is such that its *maxEdgeδ*-width is at least Δ. On the other hand, any 0-branched partitioning with maximum degree at most 3 of V can be seen as a linear ordering over the vertices of G, which implies that the *maxEdgeδ*-width of G is at least as big as cw(G). More precisely, the minimum *maxEdgeδ* width of the 0-branched partitioning trees of V with maximum degree at most 3 equals $\max\{\text{cw}(G), \Delta\}$. In general, to express the cut width of a graph, we need a more restrictive partition function.

Definition 11. Let 3-*maxEdgeδ*$_V$ be the partition function for $V \in \mathcal{V}$ that assigns the function $\max\{\mathit{Edge}\delta(V_1, V \setminus V_1), \mathit{Edge}\delta(V_2, V \setminus V_2)\}$ to any partition (V_1, V_2, V_3) of V, with $|V_3| = 1$. Let 3-*maxEdgeδ* be the partition function over $\mathcal{V}$ that assigns 3-*maxEdgeδ*$_V$ to any $V \in \mathcal{V}$.

Cut width [Thi00]: The cut width of G is at most $k \geq 1$, if and only if there is a partitioning tree (T, σ) of V with 3-*maxEdgeδ* width at most k, and (T, σ) is 0-branched. This result easily follows from the trivial correspondence between such a partitioning tree of V and an ordering of V.

4.2 Main Results

In this section, we define properties of a partition function Φ that are sufficient for Φ to admit a linear-time (in the size of the input set) algorithm that decides whether the Φ-width of any set is at most k, k being a fixed integer.

More precisely, we start by giving a set of sufficient conditions for our theorem. Then, we show that all aforementioned widths respect these conditions. This implies that the algorithm in Section 4.5 can be used to compute all the aforementioned widths, thus this algorithm generalizes the FPT-algorithms of [BK96, BT97, Thi00, BT04].

Sufficient Conditions For a Linear Time Algorithm

First, some definitions are made in this section in order to state the main theorem.

Since partitioning trees generalize the tree decomposition to any set (not only graphs), it is natural to extend the notion of *nice tree decomposition [Bod96]* to any set.

A *nice decomposition* $(T, \mathcal{X})$ of a finite set A is a $O(|A|)$-node rooted tree T, together with a family $\mathcal{X} = (X_v)_{v \in V(T)}$ of subsets of A such that, $\cup_{v \in V(T)} X_v = A$, for all $a \in A$ the set $\{v \mid a \in X_v\}$ induces a subtree of T, and for any $v \in V(T)$:

start node: v is a leaf, or

introduce node: v has a unique child u, $X_u \subset X_v$ and $|X_v| = |X_u| + 1$, or

forget node: v has a unique child u, $X_v \subset X_u$ and $|X_u| = |X_v| + 1$, or

join node: v has exactly two children u and w, and $X_v = X_u = X_w$.

The *width* of a decomposition $(T, \mathcal{X})$ is the $\max_{t \in V(T)} |X_t|$. For any $v \in V(T)$, let T_v denote the subtree of T rooted in v, and $A_v = \cup_{t \in V(T_v)} X_t$.

Let Φ be a partition function. A *nice decomposition* $(T, \mathcal{X})$ of a set A is *compatible* with Φ if:

1. there exists a function F_Φ that associates an integer $F_\Phi(x, \mathcal{P}, e)$ to any integer x, partition $\mathcal{P}$ of some subset of A and element e of A, such that, F_Φ is strictly increasing in its first coordinate, and, for any introduce node $v \in V(D)$ with child u, any partition $\mathcal{P}$ of A_v,

$$\Phi_{A_v}(\mathcal{P}) = F_\Phi(\Phi_{A_u}(\mathcal{P} \cap A_u), \mathcal{P} \cap X_v, A_v \setminus A_u).$$

2. there exists a function H_Φ that associates an integer $H_\Phi(x, y, \mathcal{P})$ to any pair of integers x, y, and partition $\mathcal{P}$ of some subset of A, such that, H_Φ is strictly increasing in its first and second coordinates, and, for any join node $v \in V(T)$ with children u and w, any partition $\mathcal{P}$ of A_v,

$$\Phi_{A_v}(\mathcal{P}) = H_\Phi(\Phi_{A_u}(\mathcal{P} \cap A_u), \Phi_{A_w}(\mathcal{P} \cap A_w), \mathcal{P} \cap X_v).$$

3. F_Φ and H_Φ have time complexity that does not depend on the size of A_v. That is, they have constant time complexity with respect to the size of A_v.

4. If $v \in V(T)$ is an introduction node with u as children, then for every partition $\mathcal{P}$ of A_v such that $\mathcal{P} \cap X_v$ is a partition of X_v with only one part, then $\Phi_{A_v}(\mathcal{P}) = \Phi_{A_u}(\mathcal{P} \cap A_u)$.

   If $v \in V(T)$ is a join node with u and w as children, then for every partition $\mathcal{P}$ of A_v such that $\mathcal{P} \cap A_w$ is a partition of A_w with only one part we have that $\Phi_{A_v}(\mathcal{P}) = \Phi_{A_u}(\mathcal{P} \cap A_u)$. Respectively, if $\mathcal{P} \cap A_u$ is a partition of A_u with only one part, then $\Phi_{A_v}(\mathcal{P}) = \Phi_{A_w}(\mathcal{P} \cap A_w)$.

Intuitively, the existence of F_Φ and H_Φ means that it is possible to quickly compute the Φ-width of some partitions $\mathcal{P}$ without knowing explicitly $\mathcal{P}$. By only knowing a restriction of $\mathcal{P}$ and the Φ-width of some restriction of $\mathcal{P}$, these restrictions being defined by the decomposition $(T, \mathcal{X})$. Moreover, the last restriction over the function Φ means that changes on the width of a partitioning tree resulted from adding elements to the partitioning tree do not propagate long. They are contained to vertices of the partitioning tree that partition X_v into at least two parts.

Main Theorem

This is the main theorem of this chapter:

Theorem 11. *Let Φ be a monotone partition function that is closed under taking subsets. Let $k, k' \geq 1$ and $q \geq 0$ be three fixed integers (q may be ∞). There exists an algorithm that solves the following problem in time linear in the size of the input set:*

input: *a finite set A and a nice decomposition $(T, \mathcal{X})$, of width at most k' for A, that is compatible with Φ,*

output: *decide if the q-branched Φ-width of A is at most k.*

Corollary 3. *Let Φ be a monotone partition function that is closed under taking subsets. Let $k \geq 1$ and $q \geq 0$ be 2 fixed integers (q may be ∞). Let $\mathcal{A}$ be a class of sets such that there exists a linear-time algorithm for computing a nice decomposition of width $O(k)$ for any set $A \in \mathcal{A}$, compatible with Φ, if it exists. There exists an algorithm that solves the following problem in time linear in the size of the input set:*

input: *a finite set A,*

output: *decide if the q-branched Φ-width of A is at most k.*

The proof of Theorem 11 is quite technical and most of the remaining part of this chapter is devoted to it. In order to explain such proof in a more didactic manner, we start with a simple algorithm to solve this problem, albeit not in linear time, and improve such algorithm in the following sections until we have a linear time algorithm.

Tractability of Width Parameters of Graphs

This section is devoted to present an application of Theorem 11 in terms of the width measures of graphs showed in Section 4.1.

Theorem 12. *Let k and q be two fixed parameters. There exists an algorithm that solves the following problem in time linear in the size of the input graph.*

input: *A graph G with maximum degree bounded by a function of q and k,*

output: *Decide if G has q-branched tree width, resp., branch width, linear width, carving width, cut width or special tree width at most k.*

Proof. In Section 4.1, we explained that several width parameters of graphs (q-branched tree width, resp., branch width, linear width, carving width, cut width or special tree width) can be defined in terms of partition functions. Therefore, the proof of Theorem 12 roughly consists in proving that the partition functions corresponding to these width parameters satisfy conditions of Theorem 11.

Bodlaender designs a linear-time algorithm that decides if the tree width of a graph G is at most k' (k' is a fixed parameter), and, if $\text{tw}(G) \leq k'$ returns a tree decomposition of width at most k' [Bod96]. Moreover, a nice tree decomposition of G can be computed in linear time from any tree decomposition of G, and without increasing its width [BK96]. Moreover, from Lemma 14, any nice tree decomposition of G can be turned into a nice decomposition of $E(G)$ with width at most dk', where d is the maximum degree of G.

Note that from Lemmas 15, 16 and 17 we have that a nice decomposition is compatible with partitioning functions for q-branched tree width, branch width, linear width, carving width, cut width and special tree width. Therefore, in order to obtain a nice decomposition of $E(G)$, we can use the algorithm in [Bod96] and Lemma 14. Since, by hypothesis, the maximum degree of G is bounded by a function of q and k, the width of the nice decomposition obtained is bounded by a function of q and k. Therefore, the hypothesis of Theorem 11 is satisfied for all the aforementioned widths, which proves Theorem 12.

□

First, the following lemma is straightforward and its proof is thus omitted.

Lemma 13. *The partition functions δ, maxδ, Edgeδ and maxEdgeδ are monotone and closed under taking subsets.*

Next three lemmas show the compatibility of some nice decomposition with the partition functions δ, *max*δ and *maxEdge*δ.

Definition 12. [BK96] A nice tree decomposition of a graph $G = (V, E)$ is a tree decomposition of G that is a nice decomposition of V.

Lemma 14. *Any nice tree decomposition $(T, \mathcal{Y})$ of a graph $G = (V, E)$ with width k can be turned into a nice decomposition $(D, \mathcal{X})$ of E. Moreover, if G has bounded maximum degree d, the width of $(D, \mathcal{X})$ is at most $d \cdot k$.*

Proof. For any $v \in V(T)$, let T_v denote the subtree of T rooted in v, and $A_v = \cup_{t \in V(T_v)} Y_t$, and let E_v be the set of edges belonging to the subgraph induced by the vertices contained in A_v that are incident to a vertex in Y_v. Any start node, resp., join node, Y_t of $(T, \mathcal{Y})$ corresponds to a start node, resp., join node, E_t of $(D, \mathcal{X})$. For any introduce node Y_t of $(T, \mathcal{Y})$, let $x \in V$ be the vertex such that $Y_t = Y_{t'} \cup \{x\}$, where t' is the single child of t in T. Let $e_1, \ldots, e_r$ be the edges that are incident to x and to some vertex in $Y_{t'}$. Then, Y_t is modified into a path of introduce nodes $E(G[Y_{t'}]) \cup \{e_1\}, E(G[Y_{t'}]) \cup \{e_1, e_2\}, \ldots, E(G[Y_{t'}]) \cup \{e_1, e_2, \ldots, e_r\}$ in $(D, \mathcal{X})$. Finally, any forget node Y_t of $(T, \mathcal{Y})$ is modified into a path of forget nodes $E(G[Y_{t'}]) \setminus \{e_1\}, E(G[Y_{t'}]) \setminus \{e_1, e_2\}, \ldots, E(G[Y_{t'}]) \setminus \{e_1, e_2, \ldots, e_r\}$ in $(D, \mathcal{X})$, where t' is the unique child of t in T, and $e_1, \ldots, e_r$ are the edges that are incident to $x = Y_{t'} \setminus Y_t$ and to no other vertex in Y_t. The obtained decomposition of E is a nice decomposition and its width (i.e. the maximum number of edges in each bag) is at most the width of the tree decomposition $(T, \mathcal{Y})$ times the maximum degree of G. □

Lemma 15. *Let G be a graph with maximum degree deg. Given a nice tree decomposition $(T, \mathcal{Y})$ of G with width at most $k' \geq 1$, a nice decomposition $(D, \mathcal{X})$ of E, compatible with the partition functions corresponding to tree width (resp., path width and special tree width) and with $\max_{t \in V(D)} |X_t| \leq k' \cdot deg$ can be computed in linear time.*

Proof. Recall that the tree width, the path width and the special tree width of a graph may be defined in terms of the partition function δ. First, let $(D, \mathcal{X})$ be the nice decomposition of E, with width at most $k' \cdot deg$, obtained from $(T, \mathcal{Y})$ as indicated in Lemma 14. We aim at proving that $(D, \mathcal{X})$ is compatible with δ. Let F_δ be defined as follows.

Definition 13. Let x be an integer, $\mathcal{P}$ be a partition of a subset E' of E and an edge $e \in E'$. Let $F_\delta(x, \mathcal{P}, e) = x + |\{v \in e \mid v \in \Delta(\mathcal{P}) \setminus \Delta(\mathcal{P} \cap (E' \setminus \{e\}))\}|$.

That is, F_δ adds to x the number of vertices incident to e that contribute to the border of the partition $\mathcal{P}$ because they are incident to e. F_δ is obviously strictly increasing in its first coordinate. Moreover, F can be computed in constant time when $|E'|$ is bounded by a constant.

For any $v \in V(D)$, let D_v denote the subtree of D rooted in v, and $A_v = \cup_{t \in V(D_v)} X_t$.

Let $v \in V(D)$ be an introduce node with child u, and let $\{e\} = X_v \setminus X_u$. Let $\mathcal{P}$ be a partition of A_v. We need to prove that $\delta_{A_v}(\mathcal{P}) = F_\delta(\delta_{A_u}(\mathcal{P} \cap A_u), \mathcal{P} \cap X_v, e)$. In other words, let us prove that $\delta_{A_v}(\mathcal{P}) = \delta_{A_u}(\mathcal{P} \cap A_u) + |\{v \in e \mid v \in \Delta(\mathcal{P} \cap X_v) \setminus \Delta((\mathcal{P} \cap X_v) \cap (X_v \setminus \{e\}))\}|$.

$\delta_{A_v}(\mathcal{P})$ is the number of vertices in the subgraph induced by the set of edges A_v that are incident to edges in different parts of $\mathcal{P}$. This set of vertices can be divided into two disjoint sets: (1) the set S_1 of vertices that are incident to two edges f and h that are different from e and that belong to different parts of $\mathcal{P}$, and (2) the set S_2 of vertices x incident to e and such that all other edges (different from e) incident to x belong to the same part of $\mathcal{P}$ that is not the part of e. S_1 is exactly the set of vertices belonging to $\Delta_{A_u}(\mathcal{P} \cap A_u)$, therefore $|S_1| = \delta_{A_u}(\mathcal{P} \cap A_u)$.

By definition of $(D, \mathcal{X})$, because it has been built from a tree decomposition, any edge of $A_u = A_v \setminus \{e\}$ that has a common end with e belongs to X_v. Therefore, any vertex in S_2 belongs to $\Delta(\mathcal{P} \cap X_v)$. It is easy to conclude that $|S_2| = |\{v \in e \mid v \in \Delta(\mathcal{P} \cap X_v) \setminus \Delta(\mathcal{P} \cap X_v \cap (X_v \setminus \{e\}))\}|$.

Therefore, the function F_δ satisfies the desired properties.

Definition 14. Let x and y be two integers, and let $\mathcal{P}$ be a partition of a subset E' of E. Let $H_\delta(x, y, \mathcal{P}) = x + y - \delta(\mathcal{P})$.

H_δ is obviously strictly increasing in its first and second coordinates. Moreover, it can be computed in constant time when $|E'|$ is bounded by a constant.

Let $v \in V(D)$ be a join node with children u and w, and let $\mathcal{P}$ be a partition of A_v, we must prove that $\delta_{A_v}(\mathcal{P}) = H_\delta(\delta_{A_u}(\mathcal{P} \cap A_u), \delta_{A_w}(\mathcal{P} \cap A_w), \mathcal{P} \cap X_v)$. That is, we prove that $\delta_{A_v}(\mathcal{P}) = \delta_{A_u}(\mathcal{P} \cap A_u) + \delta_{A_w}(\mathcal{P} \cap A_w) - \delta_{X_v}(\mathcal{P} \cap X_v)$.

First, note that $\Delta_{A_u}(\mathcal{P} \cap A_u) \cup \Delta_{A_w}(\mathcal{P} \cap A_w) \subseteq \Delta_{A_v}(\mathcal{P})$. Moreover, by definition of the nice decomposition $(D, \mathcal{X})$, an edge of $A_u \setminus X_v$ and an edge of $A_w \setminus X_v$ cannot be incident. Indeed, X_v has been built by taking all edges incident to a vertex in a bag Y of the tree decomposition $(T, \mathcal{Y})$. By the connectivity property of a tree decomposition, if a vertex x would have been incident to an edge in $A_u \setminus A_w$ and to an edge in $A_w \setminus A_u$, then $x \in Y$ which would have implied that both these edges belong to $X_v = A_u \cap A_w$, a contradiction. Therefore, $\Delta_{A_v}(\mathcal{P}) \subseteq \Delta_{A_u}(\mathcal{P} \cap A_u) \cup \Delta_{A_w}(\mathcal{P} \cap A_w)$. To conclude showing that H_δ is correct, it is sufficient to observe that $\Delta_{A_u}(\mathcal{P} \cap A_u) \cap \Delta_{A_w}(\mathcal{P} \cap A_w) = \Delta_{X_v}(\mathcal{P} \cap X_v)$.

Now, we need to show that if v is an introduction node with u as children then for all partitions $\mathcal{P}$ of A_v such that $\mathcal{P} \cap X_v$ is a partition of X_v with only one part then $\Delta_{A_v}(\mathcal{P}) = \Delta_{A_u}(\mathcal{P} \cap A_u)$.

Assume that v is an introduction node and that $A_v \setminus A_u = \{a\}$. Let $\mathcal{P}$ be a partitioning of A_v such that $\mathcal{P} \cap X_v$ is a partition of X_v with only one part.

Since $(D, \mathcal{X})$ is a nice decomposition of E, we have that a is not adjacent to any edge in $A_u \setminus X_v$. Assume that $\Delta_{A_u}(\mathcal{P} \cap A_u) < \Delta_{A_v}(\mathcal{P})$. This means that there is an extremity of a that is incident to an edge of $A_u \setminus X_v$, a contradiction. Therefore, $\Delta_{A_u}(\mathcal{P} \cap A_u) = \Delta_{A_v}(\mathcal{P})$.

Finally, we need to show that if v is a join node of $(D, \mathcal{X})$ with children u and w, then for all partitions $\mathcal{P}$ of A_v such that $\mathcal{P} \cap A_u$ is a partition of A_u with only one part then $\Delta_{A_v}(\mathcal{P}) = \Delta_{A_w}(\mathcal{P} \cap A_w)$ or $\Delta_{A_v}(\mathcal{P}) = \Delta_{A_u}(\mathcal{P} \cap A_u)$ in the case that $\mathcal{P} \cap A_w$ is a partition of A_w with only one part.

W.l.o.g. assume that $\mathcal{P} \cap A_w$ is a partition of A_w with only one part. Since, $(D, \mathcal{X})$ is a nice decomposition of E, we have that no edges in $A_u \setminus A_w$ share extremities with edges in $A_w \setminus A_u$. Moreover, $(A_u \setminus A_w) \cap (A_w \setminus A_u) = \emptyset$ and $A_u \cap A_w = X_v$.

$\Delta_{A_v}(\mathcal{P})$ is the number of vertices of G such that they are extremities for edges in different parts of $\mathcal{P}$. Since, $\mathcal{P} \cap A_w$ is a partition with only one part and the edges of $A_w \setminus A_u$ do not share any extremities with edges in $A_u \setminus A_w$. We have that $\Delta_{A_v}(\mathcal{P})$ is the number of vertices that are extremities of edges in different parts of $\mathcal{P} \cap A_u$. That is, $\Delta_{A_v}(\mathcal{P}) = \Delta_{A_u}(\mathcal{P} \cap A_u)$.

The case where $\mathcal{P} \cap A_u$ is a partition of A_u with only one part is similar and thus omitted. □

It is easy to see that the partition function *Edgeδ* behaves as the δ function but the role of vertices and edges being reversed.

First note that any nice tree decomposition $(T, \mathcal{Y})$ of G is a nice decomposition of V. To prove that $(T, \mathcal{Y})$ is compatible with the partition function *Edgeδ*, we follow the above proof of Lemma 15.

Definition 15. $F_{Edge\delta}$ and $H_{Edge\delta}$ are defined as follows:

- Let x be an integer, $\mathcal{P}$ be a partition of a subset V' of V and a vertex $v \in V'$. Then,

$$F_{Edge\delta}(x, \mathcal{P}, v) = x + |\{e \in E \mid v \in e \text{ and } e \in Edge\delta(\mathcal{P}) \setminus Edge\delta(\mathcal{P} \cap (V' \setminus \{v\}))\}|.$$

- Let x and y be two integers, and let $\mathcal{P}$ be a partition of a subset V' of V. Then, $H_{Edge\delta}(x, y, \mathcal{P}) = x + y - Edge\delta(\mathcal{P})$.

Therefore, the partition function *Edgeδ* is compatible with any nice tree decomposition. To prove the compatibility of the partition function *maxEdgeδ* with any nice tree decomposition, we use the following claim.

Let f be any partition function and let *maxf* be the partition function that associates $\max_{i \leq n} f(A_i, A \setminus A_i)$ to any partition $(A_1, \ldots, A_n)$ of A.

Claim 1. *For any partition function f compatible with a nice decomposition of some set A, the partition function maxf is also compatible.*

Remark 2. Claim 1 also serves to show that the partition functions for branch width and linear width are also compatible to a nice decomposition of the edges of a graph.

Because f is compatible with any nice decomposition of A, there exist two function F_f and H_f that satisfy the properties defining the notion of compatibility. We must have:

$$f_{A_v}(\mathcal{P}) = F_f(f_{A_u}(\mathcal{P} \cap A_u), \mathcal{P} \cap X_v, A_v \setminus A_u), \text{ and}$$
$$f_{A_v}(\mathcal{P}) = H_f(f_{A_u}(\mathcal{P} \cap A_u), f_{A_w}(\mathcal{P} \cap A_w), \mathcal{P} \cap X_v).$$

The key point is that if $\mathcal{P}$ is a bipartition of some set A, then $f_A(\mathcal{P}) = maxf_A(\mathcal{P})$. Therefore, when considering a bipartition, the functions F_{maxf} and H_{maxf} can be defined similarly to F_f and H_f. The case that $\mathcal{P}$ is not a bipartition is more technical, hence we postpone the proof of this claim until Section 4.5 after showing how the algorithm works.

Hence with Claim 1 the partition functions corresponding to the branch width and linear width (carving width and cut width) are all compatible to nice decompositions of E (V) of a graph $G = (V, E)$. This gives us the following lemmas:

Lemma 16. *Any nice decomposition $(D, \mathcal{X})$ of G is a nice decomposition of E compatible with the partition functions corresponding to branch width (resp., linear width).*

Lemma 17. *Any nice tree decomposition $(T, \mathcal{Y})$ of G is a nice decomposition of V compatible with the partition functions corresponding to carving width (resp., cut width).*

4.3 Describing Partitioning Trees in a Dynamic Manner

In this section, we show the basic idea used to compute all the aforementioned widths.

Preliminary Definitions

Definition 16. Let (T, r, σ) be a rooted partitioning tree of a set A and $A' \subseteq A$, then the *partitioning tree* (T', r', σ') *of* (T, r, σ) *restricted to* A' is the minimum subtree of T' spanning all the leaves corresponding to elements of A'. r' is the vertex of T' that is closest to r in T and the function σ' is the restriction of σ to A'.

Let $(D, \mathcal{X})$ be a nice decomposition of a set A and Φ a partitioning function of A that is compatible with $(D, \mathcal{X})$. Recall that for any $v \in V(D)$, D_v denotes the subtree of D rooted in v, and $A_v = \cup_{t \in V(D_v)} X_t$. In what follows, a *full set of partitioning trees* of a vertex $v \in V(D)$, denoted by $\mathrm{FSPT}_{\mathrm{k,q}}(v)$, is the set of all labeled q-branched partitioning trees of Φ-width at most k of A_v. If r is the root of $(D, \mathcal{X})$ then $A_r = A$, hence $\mathrm{FSPT}_{\mathrm{k,q}}(r)$ is not empty if and only if the q-branched Φ-width of A is not bigger than k.

Definition 17. A *labeled partitioning tree*, $((T, r, \sigma), \ell)$ is a partitioning tree (T, r, σ) along with a label ℓ, a function from the edges or internal vertices of T to integers, the label of a vertex (or edge) t of T is such that $\ell(t) = \Phi(\mathcal{A}_t)$, where $\mathcal{A}_t$ is the partition of A defined by t.

The role of the label is to store the values of the partitioning function Φ for fast access and update during the execution of the algorithm.

Main Idea

Let k and q be fixed integers. The main idea behind the algorithm in section Section 4.5 is to use dynamic programming to compute a full set of partitioning trees for A by using a nice decomposition $(D, \mathcal{X})$ of A. In other words, to decide if the q-branched Φ-width of A is not bigger than k. We start by computing a $\mathrm{FSPT}_{\mathrm{k,q}}(v)$ for all bags X_v where v is a leaf of D. Then, for each vertex $v \in V(D)$ such that for each child u of v we have already computed the set $\mathrm{FSPT}_{\mathrm{k,q}}(u)$, we compute $\mathrm{FSPT}_{\mathrm{k,q}}(v)$. Once $\mathrm{FSPT}_{\mathrm{k,q}}(r)$, where $r \in V(D)$ is the root of D, is computed, then we can simply test if $\mathrm{FSPT}_{\mathrm{k,q}}(r)$ is empty to decide whether the q-branched Φ-width of A is not bigger than k.

In this section we show how to compute $\mathrm{FSPT}_{\mathrm{k,q}}(v)$ for vertices of $V(D)$, for the moment, we do not focus on the complexity of this computation, rather we show the main idea behind each procedure introduced on Section 4.5. The computation of $\mathrm{FSPT}_{\mathrm{k,q}}(v)$ depends on the type of the node v. In the following subsections we show procedures for each kind of node in the nice decomposition $(D, \mathcal{X})$ (starting node, introduce node, forget node and join node).

We also state that it is possible that a full set of characteristics has infinite size, hence it is necessary to design a method to "compress" this set reducing its cardinality to something more manageable, i.e., a size given by a function bounded on k, the width of the nice decomposition and q. In Section 4.4 we show how this can be achieved.

Then, in Section 4.5, we show how to use this compression to design a linear time algorithm to decide if the Φ-width of a set A is not bigger than k, k being a fixed parameter.

Procedure Starting Node

If v is a starting node, i.e. a leaf of D. Then, procedure Starting Node consists of enumerating all q-branched partitioning trees for A_v with Φ-width at most k.

That is, $\mathrm{FSPT}_{\mathrm{k,q}}(v)$ is the set of all possible labeled q-branched partitioning trees for A_v with Φ-width at most k.

Lemma 18. *Let $(D, \mathcal{X})$ be a nice decomposition of a set A. The procedure Starting Node computes a full set of q-branched partitioning trees of Φ-width not bigger than k for a starting node v of D.*

Procedure Introduce Node

Assume that v is an introduce node of D and that we aim at computing $\mathrm{FSPT}_{\mathrm{k,q}}(v)$.

Let u be the only child of v in D and $\{a\} = X_v \setminus X_u$. Let $\mathrm{FSPT}_{\mathrm{k,q}}(u)$ be the full set of labeled q-branched partitioning trees of A_u with width at most k. The set $\mathrm{FSPT}_{\mathrm{k,q}}(v)$ is obtained from $\mathrm{FSPT}_{\mathrm{k,q}}(u)$ by applying the following procedure to every labeled partitioning tree (T_u, r_u, σ_u) in $\mathrm{FSPT}_{\mathrm{k,q}}(u)$ and to every possible execution of the step "*update T_u into T_v*".

Procedure Introduce Node. Starting with $\mathrm{FSPT}_{\mathrm{k,q}}(v) =$, for all possible choices of step "update T_u into T_v" and for all elements$(T_u, r_u, \sigma_u) \in \mathrm{FSPT}_{\mathrm{k,q}}(u)$ do the following:

update T_u into T_v: To insert a corresponding vertex to a in $((T_u, r_u, \sigma_u), \ell_u)$, choose some internal vertex v_{att} of $V(T_u)$, add a leaf v_{leaf} adjacent to v_{att}. Moreover, let $e_{new} = \{v_{att}, v_{leaf}\}$, we then proceed to subdivide e_{new} a finite number of times. Then, set $\sigma_v(v_{leaf}) = a$. Let P_{new} be the path joining v_{leaf} to v_{att}. If $r_u = v_{att}$ then r_v is one of the vertices in $V(P_{new}) \setminus \{v_{leaf}\}$, otherwise $r_v = r_u$. Note that, at this point, T_v is a partitioning tree of A_v.

update of labels of new vertex(s) and edge(s): First, let P_{new} be the path joining v_{leaf} to v_{att}. Then every internal vertex (or edge) p of P_{new} receives label $\ell_v(p) = \Phi_{A_v}(\{A_u, \{a\}\})$.

update of labels of other vertex(s) and edge(s): For all $e \in E(T_v) \setminus E(P_{new})$, let $\mathcal{T}_e$ be the partition of X_v defined by e. $\ell_v(e) \leftarrow F_\Phi(\ell_u(e), \mathcal{T}_e, a)$. For all $t \in (V(T_v) \setminus V(P_{new})) \cup \{v_{att}\}$, let $\mathcal{T}_t$ be the partition of X_v defined by t, then $\ell_v(t) \leftarrow F_\Phi(\ell_u(t), \mathcal{T}_t, a)$.

update of $\mathrm{FSPT}_{\mathrm{k,q}}(v)$: If $((T_v, r_v, \sigma_v), ell_v)$ is q-branched, for every internal vertex $t \in V(T_v)$ we have $\ell_v(t) \leq k$ and for every edge $e \in E(T_v)$ we have $\ell_v(e) \leq k$ then $\mathrm{FSPT}_{\mathrm{k,q}}(v) \leftarrow \mathrm{FSPT}_{\mathrm{k,q}}(v) \cup \{((T_v, r_v, \sigma_v), \ell_v)\}$, otherwise $\mathrm{FSPT}_{\mathrm{k,q}}(v)$ remains unchanged.

Lemma 19. *Let $(D, \mathcal{X})$ be a nice decomposition of a set A compatible with the monotone partition function Φ and let v be an introduce node of D with a child u. The procedure Introduce Node computes a full set of q-branched partitioning trees of Φ-width not bigger than k from the set $\mathrm{FSPT}_{\mathrm{k,q}}(u)$ for the node v.*

Proof. Let $\mathrm{FSPT}_{\mathrm{k,q}}(v)$ be the set computed by the procedure Introduce Node, we first show that any element $((T_v, r_v, \sigma_v), \ell_v) \in \mathrm{FSPT}_{\mathrm{k,q}}(v)$ is a q-branched partitioning tree with Φ-width not bigger than k for A_v.

Let $\mathrm{FSPT}_{\mathrm{k,q}}(u)$ be a full set of q-branched partitioning trees of Φ-width not bigger than k for the node u and let (T_u, r_u, σ_u) be any element of $\mathrm{FSPT}_{\mathrm{k,q}}(u)$. Assume that $((T_v, r_v, \sigma_v), \ell_v)$ is obtained through an execution of procedure Introduce Node on (T_u, r_u, σ_u).

From the step "update T_u into T_v", since (T_u, r_u, σ_u) is a partitioning tree for A_u and $A_v \setminus A_u = \{a\}$, taking (T_u, r_u, σ_u) adding a leaf v_{leaf} to an internal vertex v_{att} of T_u and mapping v_{leaf} to a results in a partitioning tree for A_v. Moreover, the subdivision of $\{v_{att}, v_{leaf}\}$ does not change the fact that (T_v, r_v, σ_v) is a partitioning tree for A_v. Hence, (T_v, r_v, σ_v) is a partitioning tree for A_v.

For any internal vertex (or edge) t of T_v let $\mathcal{A}_t$ be the partition of A_v that it defines. It remains to show that (T_v, r_v, σ_v) is q-branched, has Φ-width not bigger than k and that after the execution of the procedure Introduce Node all labels are correct. In other words, for all internal vertices (or edges) t of T_v, we prove that $\ell_v(t) = \Phi_{A_v}(\mathcal{A}_t)$.

In the step "update of $\mathrm{FSPT}_{\mathrm{k,q}}(v)$", $((T_v, r_v, \sigma_v), \ell_v)$ is only added to $\mathrm{FSPT}_{\mathrm{k,q}}(v)$ if it is q-branched. Then, from the fact that there are only labeled partitioning trees in $\mathrm{FSPT}_{\mathrm{k,q}}(u)$, we have that $\ell_u(t) = \Phi_{A_u}(\mathcal{A}_t \cap A_u)$ for any internal vertex (or edge) t of T_u. Moreover, we have that Φ is compatible with the nice decomposition $(D, \mathcal{X})$. Therefore, from the description of the Introduce Node procedure for every internal node t (or any edge) of T_v that is not in $P_{new} \cup \{v_{att}\}$:

$$\ell_v(t) = F_\Phi(\ell_u(t), \mathcal{T}_t, a) = F_\Phi(\Phi_{A_u}(\mathcal{A}_t \cap A_u), \mathcal{A}_t \cap X_v, a) = \Phi_{A_v}(\mathcal{A}_t).$$

Furthermore, all edges and internal vertices of P_{new} receive the label $\Phi_{A_v}(\{A_u, \{a\}\})$ from step "update of new vertex(s) and edge(s)", hence $l_v(t) = \Phi_{A_v}(\{A_u, \{a\}\})$, where t is either an internal vertex of P_{new} or an edge of P_{new}. Lastly, again from the fact that Φ is compatible with $(D, \mathcal{X})$ and from step "update of labels of other vertex(s) and edge(s):

$$\ell_v(v_{att}) = F_\Phi(\ell_u(v_{att}), \mathcal{T}_{v_{att}}, a) = F_\Phi(\Phi_{A_u}(\mathcal{A}_{v_{att}} \cap A_u), \mathcal{A}_{v_{att}} \cap X_v, a) = \Phi_{A_v}(\mathcal{A}_{v_{att}}).$$

Hence, at step "update of $\mathrm{FSPT}_{\mathrm{k,q}}(v)$", $((T_v, r_v, \sigma_v), \ell_v)$ is a labeled partitioning tree of A_v. Therefore, the step "update of $\mathrm{FSPT}_{\mathrm{k,q}}(v)$" guarantees that $((T_v, r_v, \sigma_v), \ell_v)$ is only added to $\mathrm{FSPT}_{\mathrm{k,q}}(v)$ if it is a labeled q-branched partitioning tree with Φ-width not bigger than k for A_v. Thus, any element of $\mathrm{FSPT}_{\mathrm{k,q}}(v)$ is a labeled q-branched partitioning tree with Φ-width not bigger than k for A_v.

We now show that any labeled q-branched partitioning tree with width not bigger than k for A_v is in $\mathrm{FSPT}_{\mathrm{k,q}}(v)$. Let $((T'_v, r'_v, \sigma'_v), \ell_v)$ be any q-branched partitioning tree with width not bigger than k for A_v. Let (T_u, r_u, σ_u) be (T'_v, r'_v, σ'_v) restricted to A_u. The partitioning tree (T_u, r_u, σ_u) is q-branched and its Φ-width is not bigger than k, since we only remove branches when restricting a partitioning tree and Φ is monotone. Thus, $((T_u, r_u, \sigma_u), \ell_u) \in \mathrm{FSPT}_{\mathrm{k,q}}(u)$.

Let v_{leaf} be the vertex of T'_v that corresponds to a. Since T_u is a subtree of T'_v, let $v_{att} \in V(T_u)$ be the vertex that is closest to v_{leaf} in T'_v and P_{new} be the path in T'_v joining v_{leaf} to v_{att}. Since $A_u = A_v \setminus \{a\}$, all internal vertices of P_{new} have degree two in T'_v. Hence, T'_v can be obtained from T_u in the step "update T_u into T_v" by attaching the vertex v_{leaf} to v_{att} and subdividing the edge e_{new} an amount of times equal to $|V(P_{new} \setminus \{v_{leaf}, v_{att}\})|$.

Therefore, let $((T_v, r_v, \sigma_v), \ell_v)$ be the labeled q-branched partitioning tree obtained from $((T_u, r_u, \sigma_u), \ell_u)$ with the Introduce Node procedure by adding a vertex v_{leaf} mapping a to v_{att} and subdividing $\{v_{att}, v_{leaf}\}$ an amount of times equal to $|V(P_{new} \setminus \{v_{leaf}, v_{att}\})|$. In other words, T_v and T'_v are isomorphic. Since the root of the tree does not change with the Introduce Node procedure, $r'_v = r_v = r_u$. Moreover, σ'_v and σ_v are the same, i.e. $\sigma'_v = \sigma_v = \sigma_u \cup (v_{att}, a)$. Therefore $((T'_v, r'_v, \sigma'_v), \ell'_v) = ((T_v, r_v, \sigma_v), \ell_v)$. Thus, for every q-branched partitioning tree $((T'_v, r'_v, \sigma'_v), \ell_v)$ with width not bigger than k for A_v there is an execution of process Introduce Node such that $((T'_v, r'_v, \sigma'_v), \ell_v) \in \mathrm{FSPT}_{\mathrm{k,q}}(v)$. □

Procedure Forget Node

Let v be a forget node of D, u be its child and $\mathrm{FSPT}_{\mathrm{k,q}}(u)$ be a full set of labeled q-branched partitioning trees of A_u with width at most k. Then procedure Forget Node consists of copying $\mathrm{FSPT}_{\mathrm{k,q}}(u)$. In other words, $\mathrm{FSPT}_{\mathrm{k,q}}(v) = \mathrm{FSPT}_{\mathrm{k,q}}(u)$.

Lemma 20. *Let $(D, \mathcal{X})$ be a nice decomposition of a set A compatible with the monotone partition function Φ and let v be a forget node of D with a child u. The procedure Forget Node computes a full set of q-branched partitioning trees of Φ-width not bigger than k for a forget node v of D.*

Since $A_v = A_u$ we have that $\mathrm{FSPT}_{\mathrm{k,q}}(v) = \mathrm{FSPT}_{\mathrm{k,q}}(u)$, therefore if v is a forget node this procedure produces a full set of labeled q-branched partitioning trees of A_u with width at most k for the node v.

Procedure Join Node

Let v be a join node of D, u and w its children and $\mathrm{FSPT}_{\mathrm{k,q}}(u)$ a full set of labeled q-branched partitioning trees of A_u with width at most k and $\mathrm{FSPT}_{\mathrm{k,q}}(w)$ a full set of labeled q-branched partitioning trees of A_w with width at most k.

Let $((T_u, r_u, \sigma_u), \ell_u) \in \mathrm{FSPT}_{\mathrm{k,q}}(u)$ and $((T_w, r_w, \sigma_w), \ell_w) \in \mathrm{FSPT}_{\mathrm{k,q}}(w)$. The goal is to merge "compatible" partitioning trees. We recall that, by definition of a join node of a nice decomposition, $X_u = X_w = X_v$. Hence, let $(T_u^r, r_u^r, \sigma_u^r)$ be (T_u, r_u, σ_u) restricted to X_v, i.e. the minimum subtree of T_u spanning all the leaves corresponding to elements of X_v (r_u^r is the vertex of T_u^r closest to r_u in T_u), and $(T_w^r, r_w^r, \sigma_w^r)$ be (T_w, r_w, σ_w) restricted to X_v. Then, we do the following procedure for every pair of elements of $((T_u, r_u, \sigma_u), \ell_u) \in \mathrm{FSPT}_{\mathrm{k,q}}(u)$ and $((T_w, r_w, \sigma_w), \ell_w) \in \mathrm{FSPT}_{\mathrm{k,q}}(w)$ such that T_u^r and T_w^r are isomorphic, $\sigma_u^r = \sigma_w^r$ and for every possible execution of step "Identifying T_u and T_w".

Identifying T_u and T_w: T_v is obtained by identifying all correspondent vertices of T_u^r and T_v^r. Then we remove from T_v the double edges resulting from the identification process. The root of T_v is obtained arbitrarily choosing an internal vertex of T_v. The mapping σ_v is obtained taking both mappings σ_u and σ_w, i.e. the leaves that are in T_u^r and T_w^r keep their correspondence to the elements of A. Since $X_u = X_w = X_v$, leaves that correspond to elements of X_v have the same mapping in σ_u and σ_w. Leaves that belong to $A_u \setminus A_w$ or $A_w \setminus A_u$ are only mapped by σ_u or σ_w respectively. Note that, at this point, (T_v, r_v, σ_v) is a partitioning tree of A_v.

Updating the labels: Let $(T_v^r, r_v^r, \sigma_v^r)$ be (T_v, r_v, σ_v) restricted to X_v. Let t_v be a vertex of T_v^r and $\mathcal{T}_{t_v}$ be the partition of X_v defined by t_v. Let t_u and t_w be the vertices of T_u^r and T_w^r, respectively, used to create t_v. Then, $\ell_v(t_v) \leftarrow H_\Phi(\ell_u(t_u), \ell_w(t_w), \mathcal{T}_{t_v})$. Let e_v be an edge of T_v^r and $\mathcal{T}_{e_v}$ the partition of X_v defined by e_v. Let e_u and e_w be its correspondent edges in T_u^r and T_w^r respectively. Then, $\ell_v(e_v) \leftarrow H_\Phi(\ell_u(e_u), \ell_w(e_w), \mathcal{T}_{e_v})$. For all other vertex (or edge) t of T_v: $\ell_v(t) \leftarrow \ell_u(t)$ if t is a vertex (or edge) of T_u or $\ell_v(t) \leftarrow \ell_v(t)$ if t is a vertex (or edge) of T_w.

Updating $\mathrm{FSPT}_{\mathrm{k,q}}(v)$: If $((T_v, r_v, \sigma_v), \ell_v)$ is q-branched, for every internal vertex $t \in V(T_v)$ we have $\ell_v(t) \leq k$ and for every edge $e \in E(T_v)$ we have $\ell_v(e) \leq k$ then $\mathrm{FSPT}_{\mathrm{k,q}}(v) \leftarrow \mathrm{FSPT}_{\mathrm{k,q}}(v) \cup \{((T_v, r_v, \sigma_v), \ell_v)\}$, otherwise $\mathrm{FSPT}_{\mathrm{k,q}}(v)$ remains unchanged.

Lemma 21. *Let $(D, \mathcal{X})$ be a nice decomposition of a set A compatible with the monotone partition function Φ and let v be a join node of D with a child u and a child w. The procedure Join Node computes a full set of q-branched partitioning trees of Φ-width not bigger than k from the sets $\mathrm{FSPT}_{\mathrm{k,q}}(u)$ and $\mathrm{FSPT}_{\mathrm{k,q}}(w)$ for the node v.*

Proof. Using the same scheme for the correctness of procedure Introduce Node, we first prove that all elements of $\mathrm{FSPT}_{\mathrm{k,q}}(v)$ described by the procedure Join Node are in fact labeled q-branched partitioning trees with width not bigger than k for A_v.

Let $((T_u, r_u, \sigma_u), \ell_u) \in \mathrm{FSPT}_{\mathrm{k,q}}(u)$ and $((T_w, r_w, \sigma_w), \ell_w) \in \mathrm{FSPT}_{\mathrm{k,q}}(w)$, be such that (T_u, r_u, σ_u) restricted to X_u and (T_w, r_w, σ_w) restricted to X_w satisfy the Join Node restrictions. In other words, let $(T_u^r, r_u^r, \sigma_u^r)$ and $(T_w^r, r_w^r, \sigma_w^r)$ be (T_u, r_u, σ_u) and (T_w, r_w, σ_w) restricted to X_v respectively. Then, T_u^r and T_w^r are isomorphic and $\sigma_u^r = \sigma_w^r$.

The step "Identifying T_u and T_w" can be applied on $((T_u, r_u, \sigma_u), \ell_u)$ and $((T_w, r_w, \sigma_w), \ell_w)$. Since $A_v = A_u \cup A_w$, $X_v = X_u = X_w$ and $(A_u \setminus X_u) \cap (A_w \setminus X_w) = \emptyset$, we have that (T_v, r_v, σ_v) obtained through step "Identifying T_u and T_w" is a partitioning tree of A_v.

For any internal vertex (or edge) of T_v let $\mathcal{A}_v$ be the partition of A_v that it defines. It remains to show that (T_v, r_v, σ_v) is q-branched, has Φ-width not bigger than k and that after the execution of the procedure Join Node all labels are correct. In other words, for all internal vertices (or edges) t of T_v, we prove that $\ell_v(t) = \Phi_{A_v}(\mathcal{A}_t)$.

In the step "update of $\mathrm{FSPT}_{\mathrm{k,q}}(v)$", $((T_v, r_v, \sigma_v), \ell_v)$ is only added to $\mathrm{FSPT}_{\mathrm{k,q}}(v)$ if it is q-branched. Lastly, we have that Φ is compatible with the nice decomposition $(D, \mathcal{X})$. Moreover, let $\mathcal{A}_t$, for any internal vertex (or edge) t of T_v be the partition of A_v defined by t. Then, from the fact that $((T_u, r_u, \sigma_u), \ell_u)$ and $((T_w, r_w, \sigma_w), \ell_w)$ are labeled partitioning trees, $\ell_u(t_u) = \Phi_{A_u}(\mathcal{A}_{t_u} \cap A_u)$ for any internal vertex (or edge) t_u of T_u and $\ell_w(t_w) = \Phi_{A_w}(\mathcal{A}_{t_w} \cap A_w)$ for any internal vertex (or edge) t_w of T_w. Therefore, from the description of the Join Node procedure for every internal node (or any edge) of T_v:

$$\ell_v(t) = H_\Phi(\ell_u(t), \ell_w(t), \mathcal{T}_t) = H_\Phi(\Phi_{A_u}(\mathcal{A}_t \cap A_u), \Phi_{A_w}(\mathcal{A}_t \cap A_w), \mathcal{A}_t \cap X_v) = \Phi_{A_v}(\mathcal{A}_t)$$

Hence, at step "update of $\mathrm{FSPT}_{\mathrm{k,q}}(v)$", $((T_v, r_v, \sigma_v), \ell_v)$ is a labeled partitioning tree of A_v. Therefore, the step "update of $\mathrm{FSPT}_{\mathrm{k,q}}(v)$" guarantees that $((T_v, r_v, \sigma_v), \ell_v)$ is only added to $\mathrm{FSPT}_{\mathrm{k,q}}(v)$ if it is a labeled q-branched partitioning tree with Φ-width not bigger than k for A_v.

We now show that any labeled q-branched partitioning tree with width not bigger than k for A_v is in $\mathrm{FSPT}_{\mathrm{k,q}}(v)$. Let $((T'_v, r'_v, \sigma'_v), \ell_v)$ be any q-branched partitioning tree with width not bigger than k for A_v.

Let (T_u, r_u, σ_u) be (T'_v, r'_v, σ'_v) restricted to A_u and (T_w, r_w, σ_w) be (T'_v, r'_v, σ'_v) restricted to A_w. The partitioning trees (T_u, r_u, σ_u) and (T_w, r_w, σ_w) are, by definition of restriction, q-branched and their Φ-width is not bigger than k, thus $((T_u, r_u, \sigma_u), \ell_u) \in \mathrm{FSPT}_{\mathrm{k,q}}(u)$ and $((T_w, r_w, \sigma_w), \ell_w) \in \mathrm{FSPT}_{\mathrm{k,q}}(w)$.

Let $(T_u^r, r_u^r, \sigma_u^r)$ and $(T_w^r, r_w^r, \sigma_w^r)$ be (T_u, r_u, σ_u) and (T_w, r_w, σ_w) restricted to X_v respectively. Since $X_u = X_w = X_v$ we have that $(T_u^r, r_u^r, \sigma_u^r)$ and $(T_w^r, r_w^r, \sigma_w^r)$ are such that T_u^r and T_w^r are isomorphic and $\sigma_u^r = \sigma_w^r$. Therefore, the Join Node procedure is applied to (T_u, r_u, σ_u) and (T_w, r_w, σ_w).

Therefore, let $((T_v, r_v, \sigma_v), \ell_v)$ be the labeled q-branched partitioning tree obtained from $((T_u, r_u, \sigma_u), \ell_u)$ and $((T_w, r_w, \sigma_w), \ell_w)$ with the Join Node procedure. Clearly, from the "Identifying T_u and T_v" step, $T_v = T'_v$ and $\sigma_v = \sigma'_v$. Since the procedure Join Node chooses an arbitrary internal vertex as the root of the partitioning tree, there is an

execution of this step where r_v is chosen as the root of T_v. Therefore, $((T'_v, r'_v, \sigma'_v), \ell'_v) = ((T_v, r_v, \sigma_v), \ell_v)$. □

Remarks on Width Measures

Given a graph $G = (V, E)$ and a nice decomposition $(D, \mathcal{X})$ of E or V. Then, with $\mathrm{FSPT}_{\mathrm{k,q}}(r)$ where r is the root of $(D, \mathcal{X})$, it is possible to answer if the (tree, path, branch, linear, cut, carving) width of G is less or equal than k. For that, we iterate among all $((T, r, \sigma), \ell) \in \mathrm{FSPT}_{\mathrm{k,q}}(r)$ and search for one that obeys the "structural" restrictions given by the desired width, for example, partitioning trees for the branch width are such that every internal vertex has degree three, hence it is necessary to search $\mathrm{FSPT}_{\mathrm{k,q}}(r)$ for a labeled partitioning tree that respects such restriction. In the case that there are no labeled partitioning trees with the "structural" restrictions given, then the desired width of G is bigger than k, otherwise we found a partitioning tree that proves that the desired width of G is not bigger than k.

The following sections of the chapter address the fact that the number of elements of $\mathrm{FSPT}_{\mathrm{k,q}}(v)$ is possibly infinite. In Section 4.4 we show how to store $\mathrm{FSPT}_{\mathrm{k,q}}(v)$ in an efficient manner, by only storing "compressed" representatives for each "class" of partitioning tree. Lastly, in Section 4.5, we show how to manipulate these compressed representatives in order to design an algorithm for this problem. This manipulation is a direct extension of the procedures Start Node, Introduce Node, Join Node and Forget Node when applied to "compressed" representatives of $\mathrm{FSPT}_{\mathrm{k,q}}(v)$.

4.4 Good Representatives of Partitioning Trees

In this section we outline the ideas used in order to improve the space necessary to store the q-branched partitioning trees of a node in the nice decomposition of A. In other words, in this section, we reuse a method for "compressing" path decompositions and tree decomposition in [BK96] this time applied to q-branched partitioning trees and partitioning functions. A "compression" of the set $\mathrm{FSPT}_{\mathrm{k,q}}(v)$ for a node v of the nice decomposition of A is such that the size of this compression is bounded by q, the Φ-width and the width of the nice decomposition of A. Hence, it does not depend on the size of A. Intuitively, this is achieved by keeping only "good" representatives, a.k.a. characteristics, for each q-branched partitioning tree with Φ-width at most k of A.

labeled Paths

Let Φ be a monotone partition function. As stated in Section 4.3, any partitioning tree (T, σ) can be viewed as a labeled graph, where any $v \in V(T)$ is labeled with $\Phi(\mathcal{T}_v)$ and any $e \in E(T)$ is labeled with $\Phi(\mathcal{T}_e)$. Because Φ is monotone, the label of an edge is not bigger than the label of its endpoints. In this section, we detail operations over labeled paths, that will serve as the basis of manipulating partitioning trees in the forthcoming sections.

A *labeled path* P is a path $(v_0, v_1, \ldots, v_n)$ where any vertex v_i is labeled with an integer $\ell(v_i)$, and any edge $e_i = \{v_{i-1}, v_i\}$ with an integer $\ell(e_i)$ such that the label of any edge is less or equal to the label of any of its endpoints.

A vertex $v_i \in V(P)$ or an edge $\{v_i, v_{i+1}\}$ is smaller than $v_j \in V(P)$ if $i < j$. Similarly v_i (resp., $\{v_i, v_{i+1}\}$) is smaller than $\{v_j, v_{j+1}\}$ if $i < j$. We define $\max(P)$ as the maximum integer labeling an edge or a vertex of P. Similarly, we define $\min(P)$.

Let $e = \{u, v\}$ be an edge in which we do a subdivision, let $e_1 = \{u, x\}$ and $e_2 = \{x, v\}$ be the edges in the resulting path and x be the vertex created, then $\ell(e_1) = \ell(e_2) = \ell(x) = \ell(e)$, i.e. edges and the vertex resulting from the subdivision are labeled with $\ell(e)$. An *extension* of a labeled path P is any path obtained by subdividing some edges of P an arbitrary number of times. Let P^* be an extension of P. The *originator* of an edge $e^* \in E(P^*)$ is the edge $e \in E(P)$ such that e^* is obtained in P^* by the subdivision of e. Similarly, the *originator* of a vertex $v^* \in E(P^*)$ is the correspondent vertex of P, if v^* is not the result of a subdivision, or is the edge $e \in E(P)$ that was subdivided to create v^*.

For any function $F : \mathbb{N} \to \mathbb{N}$, let $F(P)$ denote the path $(v_0, v_1, \ldots, v_n)$ where any label ℓ has been replaced by $F(\ell)$. If $P = (v_1, \ldots, v_n)$ and $Q = (w_1, \ldots, w_m)$ are two labeled paths with a common end, $v_n = w_1$, and vertex disjoint otherwise, their *concatenation* $P \odot Q$ is the labeled path $(v_1, \ldots, v_n = w_1, w_2, \ldots, w_m)$.

Contraction of a labeled path

In this section, we define an operation on labeled paths that will be widely used in the next sections. This operation is used to compress the size of a q-branched partitioning tree. For this purpose, we revisit the notion of *typical sequence* of a sequence of integers [BK96]. Roughly, the goal of the following operation is to contract some edges and vertices of P that are not "necessary" to remember the variations of the sequence $(\ell(v_0), \ell(e_1), \ell(v_1), \ldots, \ell(e_n), \ell(v_n))$.

First, let us recall the definition of the typical sequence of a sequence of integers [BK96]. Let $S = (s_i)_{i \leq 2n-1}$ be a sequence of integers. Its typical sequence $\tau(S)$ is obtained by iterating the following operations while it is possible: (1) if there is $i < |S|$ such that $s_i = s_{i+1}$, remove s_{i+1} from S, and (2) if there are $i < j - 1 < |S|$, and either, for any $i \leq k \leq j$, $s_i \leq s_k \leq s_j$, or, for any $i \leq k \leq j$, $s_i \geq s_k \geq s_j$, remove s_k from S for any $i < k < j$. Note that the order in which the operations are executed is not relevant, therefore $\tau(S)$ is uniquely defined.

The *contraction* $\mathrm{Contr}(P)$ is the path obtained from $P = (v_0, \ldots, v_n)$ with same ends by contracting some edges and vertices with the following operations until no more vertices or edges can be removed from the path, let $e_i = \{v_{i-1}, v_i\}$:

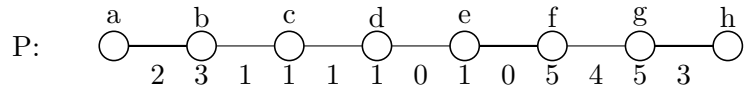
Operation 1: There exists $0 < i \leq k \leq n$ such that $\forall_{i \leq j \leq k} \ell(e_i) \leq \ell(e_j) \leq \ell(v_k)$ and $\forall_{i \leq j \leq k} \ell(e_i) \leq \ell(v_j) \leq \ell(v_k)$, then P becomes $(v_0, \ldots, v_{i-1}, v_k, \ldots, v_n)$ and $\ell(\{v_{i-1}, v_k\}) = \ell(e_i)$.

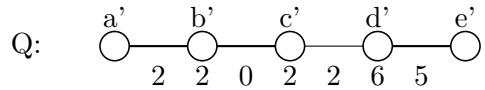
Operation 2: There exists $0 \leq i < k \leq n$ such that $\forall_{i < j \leq k} \ell(v_i) \geq \ell(e_j) \geq \ell(e_k)$ and $\forall_{i \leq j < k} \ell(v_i) \geq \ell(v_j) \geq \ell(e_k)$, then P becomes $(v_0, \ldots, v_i, v_k, \ldots, v_n)$ and $\ell(\{v_i, v_k\}) = \ell(e_k)$.

It is important to note that any $v \in V(\mathrm{Contr}(P))$ (resp. $e \in E(\mathrm{Contr}(P))$) *represents* a unique $v^* \in V(P)$ (resp. $e^* \in E(P)$), i.e. the vertex (resp. edge) in $V(P)$ (resp. $E(P)$) that originated v (resp. e) during the contraction operation. By this definition, if x represents x^* then $\ell(x) = \ell(x^*)$.

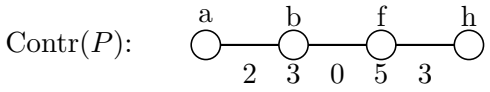
Figure 4.2 represents two labeled paths P and Q. The vertices of $\mathrm{Contr}(P)$ and $\mathrm{Contr}(Q)$ are named as the vertices they represent in P and Q. We also illustrate an extension P^* of P and an extension Q^* of Q.

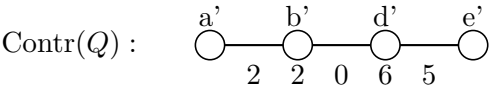
In the following, let e'_i be the edge $\{v'_{i-1}, v'_i\}$. The crucial property of $\mathrm{Contr}(P) = (v_0 = v'_0, v'_1, \ldots, v'_{p-1}, v'_p = v_n)$ is that the sequence $S' = (\ell(e'_1), \ell(v'_1), \ldots, \ell(e'_{p-1}), \ell(v'_{p-1}), \ell(e'_p))$ is "almost" the typical sequence $\tau(S)$ of $S = (\ell(e_1), \ell(v_1), \ldots, \ell(e_n))$. More precisely,



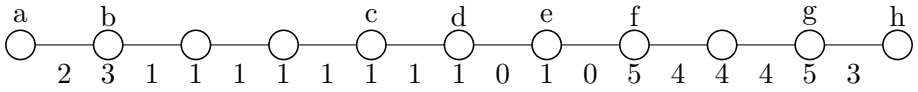


(a) paths P and Q

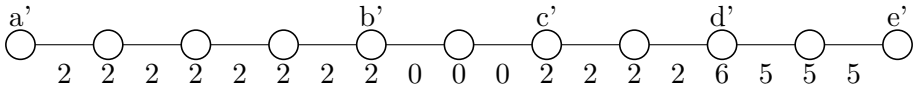




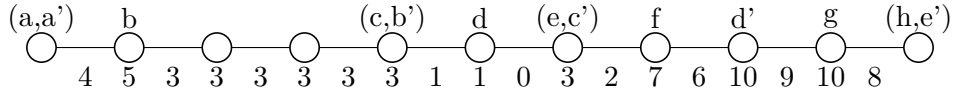
(b) contractions of P and Q



(c) an extension of P



(d) an extension of Q



(e) merging of P and Q with function +

Figure 4.2: labeled paths: contraction, extension and merging. The labels of first and last vertex are omitted, since they do not change with any operation.

- if $\ell(e'_1) \neq \ell(v'_1)$ and $\ell(e'_p) \neq \ell(v'_{p-1})$, then $S' = \tau(S)$;
- if $\ell(e'_1) = \ell(v'_1)$ and $\ell(e'_p) \neq \ell(v'_{p-1})$, then $S' = \ell(e'_1) \cdot \tau(S)$;
- if $\ell(e'_1) \neq \ell(v'_1)$ and $\ell(e'_p) = \ell(v'_{p-1})$, then $S' = \tau(S) \cdot \ell(e'_p)$;
- if $\ell(e'_1) = \ell(v'_1)$ and $\ell(e'_p) = \ell(v'_{p-1})$, then $S' = \ell(e'_1) \cdot \tau(S) \cdot \ell(e'_p)$.

Lemma 22. *Let P be a labeled path.*
1. $\min(\mathrm{Contr}(P)) = \min(P)$ *and* $\max(\mathrm{Contr}(P)) = \max(P)$.
2. *Given a labeled path P with* $\max(P) \leq k$*, then the number of edges, or size, of* $\mathrm{Contr}(P)$ *is at most* $2k + 3$.

Proof. Assume that $P = (v_1, v_2, \ldots, v_n)$. We recall that the labels of the edges are not bigger than the labels of their endpoints. Note that, the contraction operations do not change the labels of vertices or edges of P, they simply remove some vertices or edges of P. Hence, $\min(\mathrm{Contr}(P)) \geq \min(P)$ and $\max(\mathrm{Contr}(P)) \leq \max(P)$.

Assume that there exists P' such that P' is obtained after one contraction operation is applied to P and that $\max(P') < \max(P)$. W.l.o.g. assume that the contraction

operation was applied between edge the edge $e_i = \{v_{i-1}, v_i\}$ and the vertex v_j, $j > i$. In other words, $P' = (v_1, \ldots, v_{i-1}, v_j, \ldots, v_n)$. Since, $\max(P') < \max(P)$ we must have removed a vertex with a big label, That is, there is $v_l \in V(P)$, $i \leq l \leq j$, such that $\ell(v_l) > \ell(v_j)$. Therefore, by definition of a contraction operation, we are not allowed to do a contraction operation between e_i and v_j. Hence, $\min(P) = \min(P')$.

Assume that there exists P' such that P' is obtained after one contraction operation is applied to P and that $\min(P') > \min(P)$. W.l.o.g. assume that the contraction operation was applied between edge the edge $e_i = \{v_{i-1}, v_i\}$ and the vertex v_j, $j > i$. In other words, $P' = (v_1, \ldots, v_{i-1}, v_j, \ldots, v_n)$. Since, $\min(P') < \min(P)$ we must have removed an edge with small label, That is, there is $e_l \in E(P)$, $i + 1 \leq l \leq j$, such that $\ell(e_l) < \ell(e_i)$. Therefore, by definition of a contraction operation, we are not allowed to do a contraction operation between e_i and v_j. Hence, $\min(P) = \min(P')$.

Let $P = (v_0, v_1, \ldots, v_n)$, $e_i = \{v_{i-1}, v_i\}$ and $S = (\ell(e_1), \ell(v_1), \ell(e_2), \ell(v_2), \ldots, \ell(v_n))$. We have that the number of edges plus the number of vertices of $\mathrm{Contr}(P) = (v'_0, v'_1, \ldots, v'_p)$ is at most the size of $\tau(S)$ plus two, since it is possible that $\ell(e'_1) = \ell(v'_1)$ and $\ell(e'_p) = \ell(v'_{p-1})$, where $e'_i = \{v'_{i-1}, v'_i\}$.

Therefore, we have that the number of edges in $\mathrm{Contr}(P)$ is not bigger than $2k + 3$, since the number of elements in $\tau(S)$ is not bigger than $2k + 1$ [Bod96]. □

Lemma 23. *Let P and Q be two labeled paths. Let P^* be any extension of P.*
1. $\mathrm{Contr}(P^*) = \mathrm{Contr}(P) = \mathrm{Contr}(\mathrm{Contr}(P))$.
2. *Let $F : \mathbb{N} \to \mathbb{N}$ be any strictly increasing function.* $F(\mathrm{Contr}(P)) = \mathrm{Contr}(F(P))$.
3. $\mathrm{Contr}(P \odot Q) = \mathrm{Contr}(\mathrm{Contr}(P) \odot \mathrm{Contr}(Q))$.

Proof. 1. From the definition of "Contr" we have $\mathrm{Contr}(P) = \mathrm{Contr}(\mathrm{Contr}(P))$.

Let P' be obtained from P by subdividing one edge $e = \{u, v\}$ a k times resulting in $e_1 = \{u, t_1\}$, $e_2 = \{t_1, t_2\}$, $e_3 = \{t_2, t_3\}$, $\ldots$, $e_k = \{t_{k-1}, t_k\}$ and $e_{k+1} = \{t_k, v\}$. Then, for each $1 \leq i \leq k$, the labels of e_i and t_i are given by $\ell(e)$ and $\ell(e_{k+1}) = \ell(e)$, i.e. P' is the extension of P where e is subdivided k times. Then, a contraction operation can be applied between e_1 and v, $\ell(e_1) \leq \ell(t_1) = \ell(e_2) = \ell(t_2) = \cdots = \ell(t_k) = \ell(e_{k+1}) \leq \ell(v)$. The result of this contraction on path P' is P.

Then, by induction on the number of edges of P that were subdivided. Clearly, the result holds if zero edges are subdivided. Let P^* be an extension of P that subdivides at most $i \geq 0$ different edges. The above reasoning shows that it is possible to contract edges and vertices that are created through a subdivision of a single edge, i.e., if $e = \{u, v\}$ is subdivided a finite amount of times, then it is possible to apply a contraction operation on the edge created by the subdivision that has u as endpoint and v. Let P' be obtained from P^* by applying a contraction operation between v and e_1. By induction hypothesis, $\mathrm{Contr}(P') = \mathrm{Contr}(P)$, since $\mathrm{Contr}(P^*) = \mathrm{Contr}(P')$ we got the result. That is, $\mathrm{Contr}(P^*) = \mathrm{Contr}(P)$.

2. In what follows we abuse the notation of a path to include the set of edges of P, i.e. a path $P = (v'_1, \ldots, v'_r)$ becomes the path $P = (v_1, e_2, v_3, e_4, \ldots, e_{r-1}, v_{2r-1})$ where $v_{2i-1} = v'_i$ and $e_i = \{v_{i-1}, v_{i+1}\}$.

Let P be equal to $(v_1, e_2, v_3, \ldots, v_i)$ and P^f be the labeled path obtained through P by applying F, that is, $F(P)$. We show that $F(\mathrm{Contr}(P)) = \mathrm{Contr}(F(P))$ by showing that a contraction operation can be performed in P if and only if it can be performed in $F(P)$.

W.l.o.g. assume that it is possible to do a contraction operation between the edge e_i and vertex v_j, $j > i$, in P. Hence, for all e_k and v_k, $i < k < j$ we have that $\ell(e_i) \leq \ell(e_k) \leq$

$\ell(v_j)$ and $\ell(e_i) \leq \ell(v_k) \leq \ell(v_j)$. Since, F is strictly increasing, we have that for all e_k and v_k, $i < k < j$, $F(\ell(e_i)) \leq F(\ell(e_k)) \leq F(\ell(v_j))$ and $F(\ell(e_i)) \leq F(\ell(v_k)) \leq F(\ell(v_j))$. Therefore, in $F(P)$ it is possible to apply a contraction operation between e_i and v_j.

Similarly, assume that it is possible to do a contraction operation between the edge e_i and vertex v_j, $j > i$, in $F(P)$. Hence, for all e_k and v_k, $i < k < j$ we have that $F(\ell(e_i)) \leq F(\ell(e_k)) \leq F(\ell(v_j))$ and $F(\ell(e_i)) \leq F(\ell(v_k)) \leq F(\ell(v_j))$. Since, F is strictly increasing, we have that for all e_k and v_k, $i < k < j$, $\ell(e_i) \leq \ell(e_k) \leq \ell(v_j)$ and $\ell(e_i) \leq \ell(v_k) \leq \ell(v_j)$. Therefore, in P it is possible to apply a contraction operation between e_i and v_j.

Hence, the paths $\text{Contr}(P) = \{v_1^c, \ldots, v_n^c\}$ and $\text{Contr}(F(P)) = \{v_1^f, \ldots, v_n^f\}$ are composed of the same sequence of vertices and edges. That is, for all $1 \leq i \leq n$ we have that v_i^c and v_i^f represent the same vertex in P. Therefore, $F(\text{Contr}(P)) = \text{Contr}(F(P))$.

3. Comes directly from the fact that the order of contractions does not change the path obtained by applying "Contr" to a path. □

Let $P = (v_0, \ldots, v_n)$, a simple property of $\text{Contr}(P)$ is that: if v_j is a vertex with a representative in $\text{Contr}(P)$ we have that $\text{Contr}(P) = \text{Contr}((v_0, \ldots, v_j)) \odot \text{Contr}((v_j, \ldots, v_n))$.

A scheme of Lemma24 can be found in Figure 4.3.

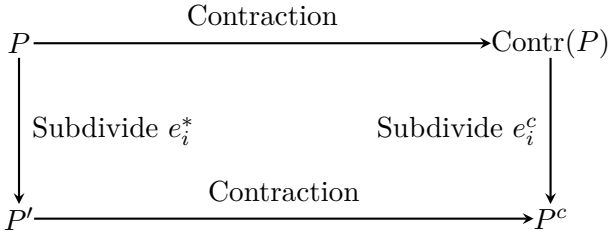


Figure 4.3: Scheme of Lemma 24.

Lemma 24. *Let $P = (v_0, \ldots, v_n)$ be a labeled path and $\text{Contr}(P) = (v_0^c, \ldots, v_p^c)$. Let $i \leq p$. Let $P^c = (v_0^c, \ldots, v_{i-1}^c, x, v_i^c, \ldots, v_p^c)$ be the extension of $\text{Contr}(P)$ obtained by subdividing once the edge $e_i^c = \{v_{i-1}^c, v_i^c\} \in \text{Contr}(P)$. Let $e_i^* = \{v_{j-1}, v_j\}$ be an edge of P represented by e_i^c, and $P' = (v_0, \ldots, v_{j-1}, y, v_j, \ldots, v_n)$ be the extension of P obtained by subdividing once e_i^*. Let $P_1^c = (v_0^c, \ldots, x)$, $P_2^c = (x, \ldots, v_p^c)$, $P_1 = (v_0, \ldots, y)$ and $P_2 = (y, \ldots, v_n)$. Then, $\text{Contr}(P_1) = \text{Contr}(P_1^c)$ and $\text{Contr}(P_2) = \text{Contr}(P_2^c)$.*

Proof. Let v_a be the vertex of P_1 represented by v_{i-1}^c in $\text{Contr}(P)$ and v_b the vertex of P_2 represented by v_i^c in $\text{Contr}(P)$. Therefore, from the definition of "Contr", $\text{Contr}((v_0, \ldots, v_a)) = \text{Contr}((v_0^c, \ldots, v_{i-1}^c))$ and $\text{Contr}((v_b \ldots, v_n)) = \text{Contr}((v_i^c, \ldots, v_p^c))$.

Since $\ell(\{v_{j-1}, y\}) = \ell(\{y, v_j\}) = \ell(\{v_{i-1}^c, x\}) = \ell(\{x, v_i^c\}) = \ell(\{v_{i-1}^c, v_i^c\}) = \ell(\{v_{j-1}, v_j\})$, we have that any contraction operation applied between v_a and $\{v_{j-1}, v_j\}$ or between $\{v_{j-1}, v_j\}$ and v_b in P can also be applied in P' between v_a and $\{v_{j-1}, y\}$ or, in the second case, between $\{y, v_j\}$ and v_b.

Therefore, $\text{Contr}((v_a, \ldots, y)) = \text{Contr}((v_{i-1}^c, x))$ and $\text{Contr}((y, \ldots, v_b)) = \text{Contr}((x, v_i^c))$. Then, from item 3 of Lemma 23 we have:

$$\begin{aligned}
\text{Contr}(P_1) &= \text{Contr}(\text{Contr}((v_0, \ldots, v_a)) \odot \text{Contr}((v_a, \ldots, y))) \\
&= \text{Contr}(\text{Contr}((v_0^c, \ldots, v_{i-1}^c)) \odot \text{Contr}((v_{i-1}^c, x))) \\
&= \text{Contr}(P_1^c)
\end{aligned}$$

The proof for $\mathrm{Contr}(P_2) = \mathrm{Contr}(P_2^c)$ is similar an thus omitted. □

Merging of labeled paths

Now, we present an operation that merges two labeled paths P and Q with common ends and vertex-disjoint otherwise. This operation is used to aid in the computation of the full set of a join node in the algorithm. The assumption that the paths have common ends is a reflection of how this operation is used by the algorithm of Section 4.5, but it is not intrinsically necessary.

A *merging* $M = (m_1, \ldots, m_k)$ of two labeled paths $P = (p_1, \ldots, p_n)$ and $Q = (q_1, \ldots, q_m)$ under a function $F : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ is a path obtained by constructing extensions $P^e = (p_1^e, \ldots, p_k^e)$ and $Q^e = (q_1^e, \ldots, q_k^e)$ of P and Q such that P^e and Q^e have the same length, $k \leq nm$. Then set $\ell(\{m_{i-1}, m_i\}) = F(\ell(\{p_{i-1}^e, p_i^e\}), \ell(\{q_{i-1}^e, q_i^e\}))$, for all $2 \leq i \leq k$, and $\ell(m_i) = F(\ell(p_i^e), \ell(q_i^e))$, for all $1 \leq i \leq k$. Note that, the maximum size of the merging, $k \leq mn$, is big enough to merge each edge on one path to each edge on the other path.

Figure 4.2 represents a merging of P and Q using the function $F : (x, y) \to x + y$. When merging P and Q, we assume they have same ends, i.e. $p_1 = q_1$ and $p_n = p_m$.

Let P and Q be two labeled paths and $M = (m_1, \ldots, m_k)$ be a merging of P and Q under a function F. Let $P^e = (p_1^e, \ldots, p_k^e)$ and $Q^e = (q_1^e, \ldots, q_k^e)$ be the extensions of P and Q used to create M.

We say that a vertex $m_i \in V(M)$ matches $p_i^e \in V(P^e)$ and $q_i^e \in V(Q^e)$. Similarly, we say that an edge $\{m_i, m_{i+1}\} \in E(M)$ matches $\{p_i^e, p_{i+1}^e\} \in E(P^e)$ and $\{q_i^e, q_{i+1}^e\} \in E(Q^e)$.

Let m be a vertex or edge of M such that m matches p^e and q^e in P^e and Q^e respectively, then $\mathrm{Orig}(m)$ is the pair (p, q) such that $p \in V(P) \cup E(P)$ is the originator, the vertex or edge of P that originated p^e, of p^e and $q \in V(Q) \cup E(Q)$ is the originator of q^e.

We now list some simple but useful properties of "Orig":

- if e is an edge of M, then $\mathrm{Orig}(e) = (p, q)$ is such that $p \in E(P)$ and $q \in E(Q)$;
- if v is a vertex of M and $\mathrm{Orig}(v) = (p, q)$ is such that $p \in E(P)$ and $q \in E(Q)$ then, let e be any edge of M with v as extremity, $\mathrm{Orig}(e) = (p, q)$. That is, if v is obtained by merging two vertices that were obtained from the subdivision of an edge, then the edges incident to v are also originated from the same edges;
- for every vertex $p \in V(P)$ (resp. $q \in V(Q)$) there is only one $v \in V(M)$ such that $\mathrm{Orig}(v) = (p, x)$ (resp. $\mathrm{Orig}(v) = (x, q)$), where x is either a vertex or edge of Q (resp. P). That is, since vertices of P and Q cannot be subdivided, they can only originate one vertex in M;
- if $M = (m_1, \ldots, m_k)$, then $\mathrm{Orig}(m_1) = (p_1, q_1)$ and $\mathrm{Orig}(m_k) = (p_n, q_m)$ where p_1 and p_n are the extremities of P and q_1 and q_m the extremities of Q.

Let P and Q be two labeled paths and M a merging of P and Q under a function F. Let P^c (resp. Q^c) be path obtained from P (resp. Q) after some, possibly zero, contraction operations are applied to P (resp. Q). Let $M^c = (m_1^c, \ldots, m_k^c)$ be a merging of P^c and Q^c under the same function F.

Roughly, we say that a merging M of P and Q *respects* a merging M^c of P^c and Q^c if the vertices and edges in M^c are obtained by matching "correspondent" vertices or edges in P and Q. Figure 4.4 shows an example of paths P, Q, P^c, Q^c, a merging M^c of P^c and Q^c and a merging M of P and Q that respects M^c.

For any vertex or edge m^c of M^c, if $\mathrm{Orig}(m^c) = (p^c, q^c)$, then let $p^c \in V(P^c) \cup E(P^c)$ and $q^c \in V(Q^c) \cup E(Q^c)$ be the representatives of $p \in V(P) \cup E(P)$ and $q \in V(Q) \cup E(Q)$ respectively. Formally, we say that M *respects* M^c if the two following conditions are met:

- for all vertices $m \in V(M)$ with $\mathrm{Orig}(m) = (p, q)$ such that p has a representative p^c in P^c, we have that there is a vertex $m^c \in V(M^c)$ such that $\mathrm{Orig}(m^c) = (p^c, q^c)$, where q^c is any vertex or edge of Q^c;
- for all vertices $m \in V(M)$ with $\mathrm{Orig}(m) = (p, q)$ such that q has a representative q^c in Q^c, we have that there is a vertex $m^c \in V(M^c)$ such that $\mathrm{Orig}(m^c) = (p^c, q^c)$, where p^c is any vertex or edge of P^c;
- for all vertices $m \in V(M)$ with $\mathrm{Orig}(m) = (p, q)$ such that such that p has a representative p^c in P^c and q has a representative q^c in Q^c, we have that there is a vertex $m^c \in V(M^c)$ such that $\mathrm{Orig}(m^c) = (p^c, q^c)$;
- for all edges $e \in E(M)$ with $\mathrm{Orig}(e) = (p, q)$ such that p has a representative p^c in P^c, we have that there is an edge $e^c \in E(M^c)$ such that $\mathrm{Orig}(e^c) = (p^c, q^c)$, where q^c is any vertex or edge of Q^c;
- for all edges $e \in E(M)$ with $\mathrm{Orig}(e) = (p, q)$ such that q has a representative q^c in Q^c, we have that there is an edge $e^c \in V(M^c)$ such that $\mathrm{Orig}(e^c) = (p^c, q^c)$, where p^c is any vertex or edge of P^c;
- for all edges $e \in E(M)$ with $\mathrm{Orig}(e) = (p, q)$ such that such that p has a representative p^c in P^c and q has a representative q^c in Q^c, we have that there is a an edge $e^c \in V(M^c)$ such that $\mathrm{Orig}(m^c) = (p^c, q^c)$.

Lemma 25. *Let $F : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ strictly increasing in both coordinates. Let P and Q be labeled paths. Let K_p and K_q be subsets of vertices of P and Q respectively. Let P^c be obtained from P by applying some contraction operations, but without contracting any vertex in K_p. Similarly, let Q^c be obtained from Q by applying some contraction operations, but without contracting any vertex in K_q.*

Let M^c be a merging of P^c and Q^c under F and M be a merging of P and Q under F that respects M^c.

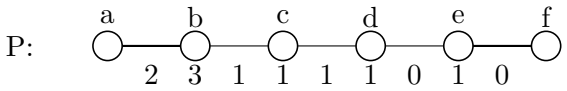
Let r be the biggest integer such that $M^c = M_1^c \odot \cdots \odot M_r^c$, where, for all $i < r$, the common end between M_i^c and M_{i+1}^c is a vertex $v_i^c \in V(M^c)$ such that $\mathrm{Orig}(v_i^c) = (p_i^c, q_i^c)$ where either the vertex represented by p_i^c is in K_p or the vertex represented by q_i^c is in K_q.

Let $M = M_1 \odot \cdots \odot M_r$, where, for all $i < r$, the common end between M_i and M_{i+1} is a vertex $v_i \in V(M)$ such that $\mathrm{Orig}(v) = (p_i, q_i)$ where either $p_i \in K_p$ or $q_i \in K_q$.

Then, for all $1 \leq i \leq r$ we have $\mathrm{Contr}(M_i) = \mathrm{Contr}(M_i^c)$.

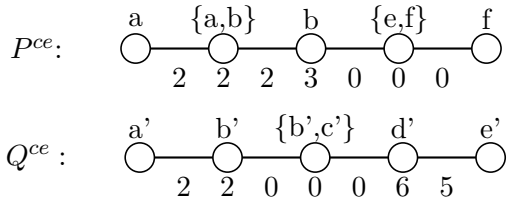
Proof. In order to prove that $\mathrm{Contr}(M_i) = \mathrm{Contr}(M_i^c)$ for all $1 \leq i \leq r$, we prove that we can obtain M_i^c from M_i with some contraction operations.

More precisely, let $M_i = (m_1, \ldots, m_h)$ and $M_i^c = (m_1^c, \ldots, m_{r'}^c)$. Consider a vertex $m^c \in V(M_i^c)$ and an edge $e^c \in E(M_i^c)$ such that m^c is an extremity of e^c. Let $\mathrm{Orig}(m^c) = (p^c, q^c)$ and let $\mathrm{Orig}(e^c) = (p'^c, q'^c)$. Since M respects M^c, there is $m \in V(M_i)$ such that $\mathrm{Orig}(m) = (p, q)$ with p and q being represented by p^c and q^c respectively and there is $e \in E(M_i)$ such that $\mathrm{Orig}(e) = (p', q')$ with p' and q' being represented by p'^c and q'^c respectively. Choose m and e such that the amount of internal vertices on the subpath of

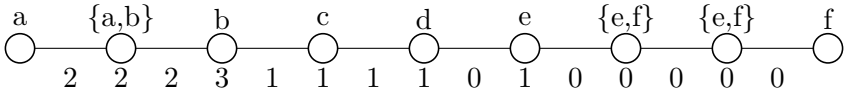


(a) paths P and Q

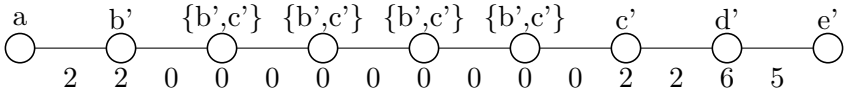
(b) P^c and Q^c



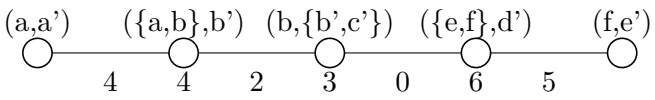
(c) extensions of P^c and Q^c



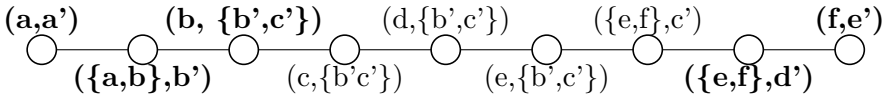
(d) extension of P



(e) extension of Q



(f) M^c: merging of P^{ce} and Q^{ce} under function +



(g) M: merging of P and Q with function + respecting M^c

Figure 4.4: A representation of a merging of P and Q that respects the merging of its contractions. Only originators of vertices are shown to avoid overloading the figure. Note that, for each originator of a vertex of M^c there is a vertex of M that has the same originator. Moreover, these originators appear in the same order in both paths M^c and M.

M from m to the extremity of e that is closest to m is the biggest. That is, if e "appears" after m in the path M, then m is the first vertex of M such that $\text{Orig}(m) = (p, q)$ and e is the last edge of M such that $\text{Orig}(e) = (p', q')$.

We show that, in M_i, if m is not an extremity of e, then it is possible do a contraction operation between m and e. In what follows assume that m is not an extremity of e in M.

Since p^c and p'^c are the originators of m^c and e^c respectively, either they are the same, meaning that m^c and e^c are originated from the subdivision of an edge of P^c, or they are different, meaning that m^c originated from a vertex (or edge) of P^c and e^c originated

from an edge (or vertex) of P^c. In the case that p^c and p'^c are different, since m^c is an extremity of e^c in M^c, it means that either p^c is a vertex that is an extremity of p'^c in P^c or that p^c is an edge with p'^c as extremity in P^c. Hence, if $p^c \neq p'^c$, then, since p^c represents p and p'^c represents p', it is possible to contract all vertices and edges between p and p' in P.

With a similar reasoning we have that, if $q^c \neq q'^c$, then it is possible to contract all vertices and edges between q and q' in Q.

There are four cases to consider: $p^c \neq p'^c$ and $q^c \neq q'^c$; $p^c = p'^c$ and $q^c \neq q'^c$; $p^c \neq p'^c$ and $q^c = q'^c$; $p^c = p'^c$ and $q^c = q'^c$.

1. $p^c \neq p'^c$ and $q^c \neq q'^c$: then, in P (resp. in Q), it is possible to contract all vertices between p and p' (resp. q and q'). Then, from the fact that H is strictly increasing in both coordinates, it is possible to contract all vertices between m and e in M.

   Formally, w.l.o.g. assume that $p^c \neq p'^c$ be such that $\ell(p^c) \geq \ell(p'^c)$ and that $q^c \neq q'^c$ be such that $\ell(q^c) \geq q'^c$. Then, in P, all vertices or edges x^p between p and p' are such that $\ell(p') \leq \ell(x^p) \leq \ell(p)$ and, in Q, all vertices or edge x^q between q and q' are such that $\ell(q') \leq \ell(x^q) \leq \ell(q)$. Therefore, since H is strictly increasing in both coordinates, $H(\ell(p'), \ell(q')) \leq H(\ell(x^p), \ell(x^q)) \leq H(\ell(p), \ell(q))$ for all vertices or edges x^p and x^q that are between p and p' in P and between q and q' in Q respectively. Hence, in M, it is possible to contract all vertices between m and e.

2. $p^c = p'^c$ and $q^c \neq q'^c$: then, in Q, it is possible to contract all vertices between q and q'. Since $p^c = p'^c$, it means that $p = p'$, therefore the vertex m, the edge e an all vertices or edge in between m and e are obtained from a subdivision of the same edge p in P. Then, from the fact that H is strictly increasing in both coordinates and the fact the first coordinate of H is the same when applying to all vertices and edges on the path between m and e, it is possible to contract all vertices between m and e in M.

   Formally, w.l.o.g. assume that $p^c = p'^c$ and that $q^c \neq q'^c$ be such that $\ell(q^c) \geq q'^c$. Then, in Q, all vertices or edge x^q between q and q' are such that $\ell(q') \leq \ell(x^q) \leq \ell(q)$. Therefore, since H is strictly increasing in both coordinates, $H(\ell(p'), \ell(q')) \leq H(\ell(p'), \ell(x^q)) \leq H(\ell(p'), \ell(q))$ for all vertices or edges x^q that are between q and q' in Q. Hence, in M, it is possible to contract all vertices between m and e.

3. $p^c \neq p'^c$ and $q^c = q'^c$: this case is similar to the case ($p^c = p'^c$ and $q^c \neq q'^c$) and thus omitted.

4. $p^c = p'^c$ and $q^c = q'^c$: this means that m, e and all vertices and edges of M between m and e where obtained from the subdivision of an edge p in P and an edge q in Q, hence they all have the same label which is given by $H(\ell(p), \ell(q))$. Therefore, it is possible to do a contraction operation between m and e in M.

Then, let M'_i be obtained from M_i by applying the above reasoning for each edge $e^c \in E(M^c)$ and with both its extremities. The resulting path M'_i is M_i^c. Since $\text{Contr}(M'_i) = \text{Contr}(M_i^c)$ and $\text{Contr}(M_i) = \text{Contr}(M'_i)$, we have the result. □

Characteristics - Good Representatives

In this section we introduce the notion of "characteristic" of a partitioning tree. The idea behind a characteristic is that, in order to compute a partitioning tree for a node v in

the nice decomposition $(D, \mathcal{X})$, the "only" important information is given by elements of X_v and the "structure" of the partitioning tree. Hence, it is possible to "forget" elements of $A_v \setminus X_v$ from the partitioning trees for v.

Restriction of a partitioning tree

Let (T', r', σ') be partitioning tree of a set A and Φ_A a monotone partition function over A. We assume that T' is not restricted to an edge, and r' is not a leaf of T' to avoid unnecessary simple cases. Therefore, the corpse $\mathrm{cp}(T')$ (T' without its leaves) can be rooted in r'. Let $B \subseteq A$, the *restriction* $\mathrm{Char}((T', r', \sigma'), B)$ *of* (T', r', σ') *to* B is composed of the following structures:

- a rooted partitioning tree (T, r, σ) of B;
- an integer dist used to remember the number of branching nodes between r' and r;
- a subset K of vertices of T used to remember the set of branching nodes and parents of leaves; and
- labeling functions:
  - $\ell : V(\mathrm{cp}(T)) \cup E(T) \to \mathbb{N}$ used to remember the width of the partitioning tree;
  - $\mathit{out} : V(\mathrm{cp}(T)) \to \mathbb{N}$ used to remember the number of branching nodes that were "forgotten";
  - $\mathit{branch} : V(\mathrm{cp}(T)) \to \{0, 1\}$ used to remember if the node in question is a branching node;
  - $\mathit{father} : V(\mathrm{cp}(T)) \to \{0, 1\}$ used to remember if the node in question is not yet a branching node, but can become one if another child is added to it.

$\mathrm{Char}((T', r', \sigma'), B)$ is computed as follows.
1. Let T be the smallest subtree spanning the leaves of T' that map elements of B. Let r be the vertex of T that is closest to r' in T'. From now on, T is rooted in r. For any leaf f of T, let $\sigma(f) = \sigma'(f)$.
2. Let dist be the number of branching nodes on the path between r' and r in $\mathrm{cp}(T') \setminus r$. If $r \neq r'$, then dist $= 0$.
3. Let K be the set of vertices of T that are either a leaf of T, or the parent of a leaf of T, or a branching node of $\mathrm{cp}(T')$ in $V(T)$ (rooted in r'), or a branching node of (T, r). The last condition seems redundant, but is necessary in case the root of the tree changes, that is, in the case that $r' \neq r$.
4. For any vertex v of $V(\mathrm{cp}(T))$, $\mathit{branch}(v) = 1$ if v is a branching node of $\mathrm{cp}(T')$, and $\mathit{branch}(v) = 0$ otherwise.

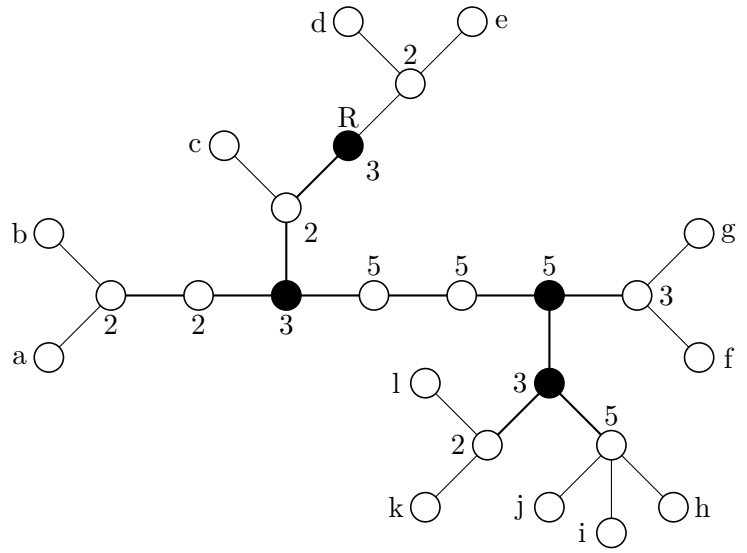
For all $v \in V(T)$, let P_v be the set of paths between v and a leaf in $T' \setminus T$ all internal vertices of which are different from r and in $T' \setminus T$.

Let $\mathit{out}(v)$ be the maximum number of branching nodes that are internal to a path in P_v.

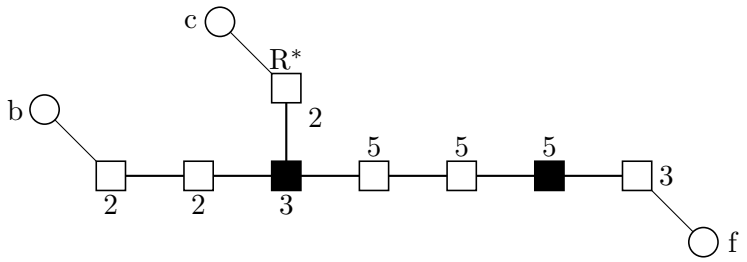
Let $\mathit{father}(v) = 1$ if v has a child that is not a leaf in T', otherwise let $\mathit{father}(v) = 0$.
5. Any internal vertex $v \in V(T)$ (resp., any edge $e \in E(T)$): $\ell(v) = \Phi_A(\mathcal{T}_v)$ (resp., $\ell(e) = \Phi_A(\mathcal{T}_e)$). Where $\mathcal{T}_v$ ($\mathcal{T}_e$) denotes the partition of A defined by v (e) in T'.
6. Then, in T, for any two vertices v, w in K such that no internal vertices of the path P between v and w are in K, replace P by $\mathrm{Contr}(P)$.

Remark 3. If $q = \infty$, we don't need to take the variables dist, *out*, *branch* and *father* into account. More precisely, the items 2, and 4 of the procedure Char can be removed and K is the set of vertices of T that are either a leaf of T, the parent of a leaf of T, or a branching node of (T, r).

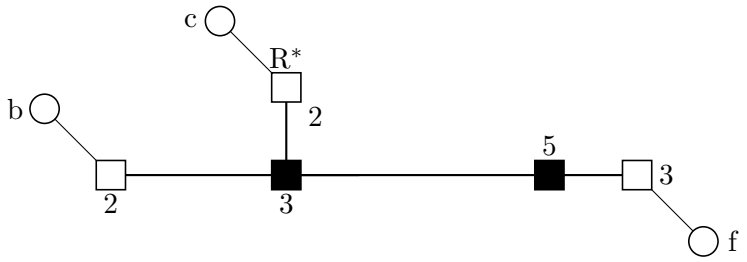
We denote by $C = \mathrm{Char}((T, r, \sigma), B)$ the restriction of (T, r, σ) to B. Figure 4.5 illustrates the process Char when applied to a labeled partitioning tree of a set A.



(a) labeled partitioning tree



(b) before step 6



(c) characteristic after contraction of paths, labels other than ℓ are omitted

Figure 4.5: Partitioning tree (T, R, σ) of $\{a, b, \ldots, l\}$ and an execution of $\mathrm{Char}((T, R, \sigma), B)$ where $B = \{b, c, f\}$. Black nodes represent the branching nodes of T.

The key point for the understanding of the relationship between the partitioning tree (T', r', σ') of A and its restriction $((T, r, \sigma), \ell, K, \mathrm{dist}, \mathit{out}, \mathit{branch}, \mathit{father})$ to B is based on the following. Any vertex of K represents a specific vertex of T' that is either a leaf of T' that maps an element of B, or the parent of such a leaf in T', or a branching node of $\mathrm{cp}(T')$ or a vertex of T' that defines a partition of B with at least three parts. Any path

P between two vertices v, w in K such that no internal vertices of P between v and w are in K, represents a path $P(v, w)$ in T' the internal vertices of which have degree two in T'. Moreover, by definition of the operation $P = \text{Contr}(P(v, w))$, any vertex (resp., edge) of P represents a specific vertex (resp., edge) of T'. Beside, by Lemma 22, the maximum (minimum) label over the vertices and edges of T is the maximum (minimum) label over the vertices and edges of T'. In particular, if (T', r', σ') has Φ-width at most k, then $\ell(v) \leq k$ and $\ell(e) \leq k$ for any $v \in V(T)$ and $e \in E(T)$.

Let the *br-height* of v, denoted by $\text{brheight}_T(v)$, in $\text{cp}(T)$ be the maximum number of branching nodes in a path from v to a leaf of the subtree of $\text{cp}(T')$ rooted in v, i.e. v union the component of $T' \setminus v$ that does not contain r'. That is, $\text{brheight}_T(v) = 0$ if v is a leaf of T, otherwise $\text{brheight}_T(v) = \max\{out(v), \max_{u \text{ child of } v}\{\text{brheight}(u)\}\} + branch(v)$. Lemma 26 shows that if a partitioning tree is q-branched then the brheight of the root of the restriction of this partitioning tree is not bigger than q.

Lemma 26. *If (T', r', σ') is a q-branched partitioning tree for A, then* $\text{Char}((T', r', \sigma'), B) = ((T, r, \sigma), \ell, K, \text{dist}, out, branch, father)$ *is such that* $\text{brheight}_T(r) + \text{dist} \leq q$

Proof. Let T'' be the subtree of T' obtained by taking the union of all paths P in T' such that one extremity of P is r' and P passes through r. Clearly T (before Step 6) is a subtree of T''. Moreover, let $r'' = r'$ and σ'' be the restriction of σ' over the leaves of T''.

Then, the labels dist, *out* and *branch* are sufficient to remember if (T'', r'', σ'') is q-branched. By the definition of br-height, (T'', r'', σ'') is q-branched if and only if the br-height of r'' is at most q.

If v is a leaf of T, it is a leaf of T'', then $\text{brheight}_T(v) = 0$. Otherwise, the br-height of $v \in V(T)$ is given by $\max\{out(v), height\} + branch(v)$, where *height* is the maximum of the br-height among the children of v.

In particular, if (T'', r'', σ'') is q branched, $out(v) \leq q$ for any $v \in K$. Finally, the $\text{brheight}_T(r'') = \text{brheight}_T(r) + \text{dist} \leq q$, since (T'', r'', σ'') is q branched. □

Characteristic of *A* restricted to *B*

Let $((T, r, \sigma), \ell, K, \text{dist}, out, branch, father)$ be such that (T, r, σ) is a rooted partitioning tree of $B \subseteq A$, $\ell : V(\text{cp}(T)) \cup E(T) \to \mathbb{N}$, $K \subseteq V(T)$ that contains at least all leaves, parents of leaves, the root and vertices with degree at least three of T, $\text{dist} \in \mathbb{N}$, $out : V(\text{cp}(T)) \to \mathbb{N}$, $branch : V(\text{cp}(T)) \to \{0, 1\}$, $father : V(\text{cp}(T)) \to \{0, 1\}$ and for any $v, w \in K$ such that no internal vertices of the path P between v and w are in K, $P = \text{Contr}(Q)$ (i.e. P results from some contraction).

Definition 18. $((T, r, \sigma), \ell, K, \text{dist}, out, branch, father)$ is a *characteristic of A restricted to B* if it exists a partitioning tree (T', r', σ') of A, such that $((T, r, \sigma), \ell, K, \text{dist}, out, branch, father) = \text{Char}((T', r', \sigma'), B)$. $((T, r, \sigma), \ell, K, \text{dist}, out, branch, father)$ is a (k, q)-*characteristic* of A restricted to B if, moreover, $\ell : V(\text{cp}(T)) \cup E(T) \to [0, k]$, and $\text{dist} + \text{brheight}_T(r) \leq q$.

Lemma 27. *If it exists a q-branched partitioning tree (T', r', σ') of A with Φ-width at most k, such that $C = \text{Char}((T', r', \sigma'), B)$, then $C = ((T, r, \sigma), \ell, K, \text{dist}, out, branch, father)$ is a (k, q)-characteristic of A restricted to B*

Proof. This is a direct consequence of Definition 18 and Lemma 26. □

Definition 19. The size of a (k, q)-characteristic of A restricted to B, $((T, r, \sigma), \ell, K, \text{dist}, out, branch, father)$, is given by the expression $|V(T)| + |K|$.

Lemma 28. *If $q < \infty$, then the number of (k,q)-characteristic of A restricted to B, with $|B| = b$, is bounded by a function $f(k,q,b) = O((60kqb)^{45kqb})$, otherwise the number of (k,∞)-characteristic of A restricted to B is bounded by a function $f(k,b) = O((15kb)^{45kb})$.*

Moreover, if $q < \infty$, then the size of a (k,q)-characteristic of A restricted to B is bounded by a function $f'(k,q,b) = O(kqb)$, otherwise the size of a (k,∞)-characteristic of A restricted to B is bounded by a function $f'(k,b) = O(kb)$.

Proof. Let $((T,r,\sigma),\ell,K,\text{dist},\mathit{out},\mathit{branch},\mathit{father})$ be a (k,q)-characteristic of A restricted to B. T is a tree with b leaves.

If $q < \infty$, i.e., q is bounded.

Since $(T, r\sigma)$ is q-branched, for each leaf there are at most q branching nodes between r and this leaf. Therefore, $|K| \leq bq + 2b + 1$.

Any path between two vertices in K (that does not contain any other vertex in K) has at most $2k+2$ internal vertices (Lemma 22). Hence, among all these paths there are at most $(bq + 2b)(2k + 2)$ vertices.

Let $n = (bq + 2b)(2k + 3) + 1 \leq 15kqb$. The number of vertices in T is obtained by taking all vertices in K and all vertices that lies on a path between two vertices of K (that does not contain any other vertex in K). Thus, $|V(T)| \leq n$.

We can bound the maximum number of non-isomorphic trees on n vertices by n^{n-2} [Cay89]. Let $\mathrm{T}(n) = \sum_{i=1}^{n} n^{n-2}$. The value $\mathrm{T}(n)$ is the number of different trees that can be the "base" of the characteristic. There are at most n vertices that can be the root of the tree and at most $b!$ (factorial) ways of mapping leaves of T to elements of B. Moreover, for each of these trees we can assign values for all the other variables, i.e., *branch*, *out*, dist, *father*, ℓ.

We have that dist $\leq q$ and for any vertex $v \in V(\mathrm{cp}(T))$ and edge $e \in E(T)$, $\ell(v) \leq k$, $\ell(e) \leq k$, $\mathit{branch}(v) \leq 1$, $\mathit{out}(v) \leq q$ and $\mathit{father}(v) \leq 1$.

Then, the number of different characteristics is bounded by:

$$\mathrm{T}(n)n(b!)q2^n q^n 2^n k^{2n-1} = 4^n q^{n+1} k^{2n-1}(b!)n\mathrm{T}(n).$$

Hence, the number of (k,q)-characteristic of A restricted to B is given by the function

$$f(k,q,|B|) = 4^n q^{n+1} k^{2n-1}(b!)n\mathrm{T}(n).$$

Since $n = O(15kqb)$, $q^{n+1} = O(q^{2n})$, $b! = O(b^{2n})$ and $\mathrm{T}(n) = O(n^n)$ with a coarse analysis we have that $f(k,q,b) = O((60kqb)^{45kqb})$.

To measure the size of one (k,q)-characteristic of A restricted to B, we can use the function $f'(k,q,b) \leq 2n$, since $|K| \leq n$.

If $q = \infty$, then $|K| \leq 3b$. Let $n' = 3b(2k+3) < 15kb$ and $\mathrm{T}'(n') = \sum_{i=1}^{n} n'^{n'-2}$. Using the same reasoning as for the proof of the case $q < \infty$, we have that the number of (k,∞)-characteristic of A restricted to B is given by a function:

$$f(k,b) = k^{2n'-1}(b!)\mathrm{T}'(n').$$

Since $b! = O(b^b) = O(b^{kb})$, $n' < 15kb$ and $\mathrm{T}'(n') = O(n'^{n'})$ with a coarse analysis we have that $f(k,b) = O((15kb)^{45kb})$

To measure the size of one (k,q)-characteristic of A restricted to B, we can use the function $f'(k,b) \leq n' + 3b$, since $|K| \leq 3b$. □

Definition 20. A set F of (k,q)-characteristics of A restricted to B is *full* if for all q-branched partitioning tree (T,r,σ) of A with Φ-width at most k, then $\mathrm{Char}((T,r,\sigma),B) \in F$.

4.5 Algorithm Using Characteristic

This section is devoted to the presentation of Procedures used in the decision algorithm for the q-branched Φ-width of a set A. Notations are those defined in Sections 4.1, 4.2 for Theorem 11.

Let $(D, \mathcal{X})$ be a nice decomposition for A that is compatible with a monotone partition function Φ, such that $\max_{t \in V(D)} |X_t| \leq k'$. Recall that for any $v \in V(D)$, D_v denotes the subtree of D rooted in v, and $A_v = \cup_{t \in V(D_v)} X_t$.

This section presents procedures that compute a full set $\mathrm{FSC_{k,q}}(t)$ of (k,q)-characteristics of A_t restricted to X_t, for any $t \in V(T)$. The algorithm proceeds by dynamic programming from the leaves of D to its root.

Each procedure presented in this section takes as input a node $v \in V(D)$ and, for each u that is a child of v, the sets $\mathrm{FSC_{k,q}}(u)$. The output of each procedure is $\mathrm{FSC_{k,q}}(v)$.

Procedure *StartNode*

If v is a leaf, i.e. a start node of D, $A_v = X_v$, and $|X_v| \leq k'$. $\mathrm{FSC_{k,q}}(v)$ consists of all (k,q)-characteristics of X_v.

Procedure *StartNode* enumerates all (k,q)-characteristics of A_v restricted to X_v.

Trivially, the following statement holds:

Lemma 29. *Procedure* StartNode *computes a full set of (k,q)-characteristics of A_v restricted to X_v.*

Next lemma shows that the complexity of procedure *StartNode* does not depend on $|A|$.

Theorem 30. *Procedure* StartNode *has constant time complexity. That is, if $q < \infty$ then procedure* StartNode *computes $\mathrm{FSC_{k,q}}(v)$ in time $O((60kqk')^{46kqk'})$, otherwise procedure* StartNode *computes $\mathrm{FSC_{k,q}}(v)$ in time $O((15kk')^{46qk'})$.*

Proof. By Lemma 28, $|\mathrm{FSC_{k,q}}(v)|$ is bounded by the function $f(k,q,k')$, if $q < \infty$, or by the function $f(k,k')$, if $q = \infty$. Moreover, each element of $\mathrm{FSC_{k,q}}(v)$ has a size bounded by the function by the function $f'(k,q,k')$, if $q < \infty$, or by the function $f'(k,k')$, if $q = \infty$.

Therefore, the amount of memory needed to store $\mathrm{FSC_{k,q}}(v)$ has size bounded by $f(k,q,k') \cdot f'(k,q,k')$ in the case that $q < \infty$ or $f(k,k') \cdot f'(k,k')$ in the case that $q = \infty$.

Since $f(k,q,k') = O((60kqk')^{45kqk'})$ and $f'(k,q,k') = O(kqk')$ we have that the characteristics in $\mathrm{FSC_{k,q}}(v)$ can be enumerated in $O((60kqk')^{46kqk'})$ in the case that $q < \infty$.

Since $f(k,k') = O((15kk')^{45kqk'})$ and $f'(k,k') = O(kk')$ we have that the characteristics in $\mathrm{FSC_{k,q}}(v)$ can be enumerated in $O((15kk')^{46kk'})$ in the case that $q = \infty$. □

Procedure *IntroduceNode*

Let v be an introduce node of D, u its child, and $\{a\} = X_v \setminus X_u$. Let $\mathrm{FSC_{k,q}}(u)$ be a full set of (k,q)-characteristics of A_u restricted to X_u. For each characteristic $C_u = ((T_u, r_u, \sigma_u), \ell_u, K_u, \mathrm{dist}_u, out_u, branch_u, father_u) \in \mathrm{FSC_{k,q}}(u)$, Procedure *IntroduceNode* proceeds as follows, repeating the five steps below, for any possible execution of Step 1. Roughly, it tries all possible ways to insert a into C_u obtaining C_v.

1. *update of T_u into T_v:*

There are two ways of inserting a in C_u. Either choose an internal vertex v_{att} of $V(T_u)$, add a leaf v_{leaf} adjacent to v_{att} (*Case 1*), or choose an edge $f = \{v_{top}, v_{bottom}\}$ (with v_{top} closer to the root r_u than v_{bottom}), subdivide it into $e_{top} = \{v_{top}, v_{att}\}$ and $e_{bottom} = \{v_{att}, v_{bottom}\}$ and add a new leaf v_{leaf} adjacent to the new node v_{att} (*Case 2*). In both cases, σ_v keeps the same mapping as σ_u for leaves that are not v_{leaf}. We set $\sigma_v(v_{leaf}) = a$. Note that, now, T_v is a partitioning tree of X_v.

2. *update of labels of new vertex(ices) and edge(s):*

   In both cases of Step 1, $e_{new} = \{v_{leaf}, v_{att}\}$ receives label $\ell_v(e_{new}) \leftarrow \Phi_{A_v}(\{A_u, \{a\}\})$.

   In Case 2 of Step 1, v_{att} is a new vertex, then $\ell_v(v_{att}) = \ell_v(e_{top}) = \ell_v(e_{bottom}) \leftarrow \ell_u(f)$, and $out_v(v_{att}) = branch_v(v_{att}) \leftarrow 0$. If v_{bottom} is not a leaf of T_u, then $father_v(v_{att}) \leftarrow 1$. Otherwise, $father_v(v_{att}) \leftarrow 0$.

3. *update of labels of vertex(ices) and edge(s):*

   For each $e \in E(T_v)$, $e \neq e_{new}$, let $\mathcal{T}_e$ be the partition of X_v defined by e. $\ell_v(e) \leftarrow F_\Phi(\ell_u(e), \mathcal{T}_e, a)$.

   For each $t \in V(\mathrm{cp}(T_v))$, let $\mathcal{T}_t$ be the partition of X_v defined by t. $\ell_v(t) \leftarrow F_\Phi(\ell_u(t), \mathcal{T}_t, a)$.

   $\mathrm{dist}_v \leftarrow \mathrm{dist}_u$ and, for all internal vertex x of T_v, $out_v(x) \leftarrow out_u(x)$, $branch_v(x) \leftarrow branch_u(x)$ and $father_v(x) \leftarrow father_u(x)$.

   In Case 2 of Step 1, $father_v(v_{top}) \leftarrow 1$.

4. *creation of a new branching node:*

   In Case 1 of Step 1, $K_v \leftarrow K_u \cup \{v_{att}, v_{leaf}\}$.

   In Case 2 of Step 1, if v_{bottom} is the only child of v_{top} in T_u and $father_u(v_{top}) = 0$ (this implies that v_{top} belongs to K_u only because it is the parent of a single leaf), then $K_v \leftarrow (K_u \cup \{v_{att}, v_{leaf}\}) \setminus \{v_{top}\}$. Otherwise, $K_v \leftarrow K_u \cup \{v_{att}, v_{leaf}\}$.

   In Case 2 of Step 1, if v_{bottom} is a leaf of T_u and $father_u(v_{top}) = 1$, then

   $$branch_v(v_{top}) \leftarrow 1.$$

5. *contraction of paths:*

   $\forall x, y \in K_v$ and path P between x and y such that no internal vertices of P are in K_v, $P \leftarrow \mathrm{Contr}(P)$.

6. *update of* $\mathrm{FSC_{k,q}}(v)$*:*

   $\mathrm{brheight}_T(r_v)$ is computable thanks to out_v and $branch_v$ as seen in Lemma 26. If $\mathrm{dist}_v + \mathrm{brheight}_T(r_v) \leq q$ and $\ell_v(t) \leq k$ for any internal vertex $t \in V(T_v)$, and $\ell_v(e) \leq k$ for any edge $e \in E(T_v)$, then $\mathrm{FSC_{k,q}}(v) \leftarrow \mathrm{FSC_{k,q}}(v) \cup \{C_v\}$.

The rest of this section is dedicated to show that procedure *IntroduceNode* computes $\mathrm{FSC_{k,q}}(v)$ in constant time. We start by showing that the Procedure *IntroduceNode* computes a full set of (k, q)-characteristics of A_v restricted to X_v, then we analyze its complexity.

Lemma 31. *Procedure* IntroduceNode *computes a full set of* (k, q)*-characteristics of* A_v *restricted to* X_v*.*

Since Φ is closed under taking subset, A_v admits a q-branched partitioning tree with Φ-width at most k only if A_u does. Therefore, we can assume that $\mathrm{FSC}_{\mathrm{k,q}}(u) \neq \emptyset$, otherwise, A_v does not admit a q-branched partitioning tree with Φ-width at most k, and $\mathrm{FSC}_{\mathrm{k,q}}(v) = \emptyset$. The proof of Lemma 31 is twofold. We first prove that the set $\mathrm{FSC}_{\mathrm{k,q}}(v)$ returned by Procedure *IntroduceNode* is a set of characteristics of A_v restricted to X_v in Lemma 32, then we prove it is full in Lemma 33.

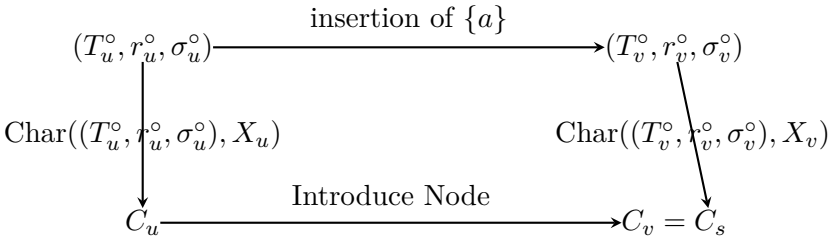


Figure 4.6: Scheme of proof of Lemma 31.

To prove that the set $\mathrm{FSC}_{\mathrm{k,q}}(v)$ is a set of characteristics, we start from a partitioning tree $(T_u^\circ, r_u^\circ, \sigma_u^\circ)$ of A_u and its corresponding characteristic C_u. The insertion of a into C_u, from the *IntroduceNode*, results in C_v. By inserting a into $(T_u^\circ, r_u^\circ, \sigma_u^\circ)$ mimicking the insertion of a into C_u we obtain $(T_v^\circ, r_v^\circ, \sigma_v^\circ)$. Then, we show that $C_v = \mathrm{Char}((T_v^\circ, r_v^\circ, \sigma_v^\circ), X_v)$. A scheme can be found in Figure 4.6.

Lemma 32. *For all $C_v \in \mathrm{FSC}_{\mathrm{k,q}}(v)$, we have that C_v is (k,q)-characteristic of A_v restricted to X_v.*

Proof. We introduce some notation in order to prove the lemma.

Let $C_u = ((T_u, r_u, \sigma_u), \ell_u, K_u, \mathrm{dist}_u, \mathit{out}_u, \mathit{branch}_u, \mathit{father}_u) \in \mathrm{FSC}_{\mathrm{k,q}}(u)$ be a characteristic used by procedure *IntroduceNode* to obtain C_v. That is, $C_v = ((T_v, r_v, \sigma_v), \ell_v, K_v, \mathrm{dist}_v, \mathit{out}_v, \mathit{branch}_v, \mathit{father}_v) \in \mathrm{FSC}_{\mathrm{k,q}}(v)$ is an element of $\mathrm{FSC}_{\mathrm{k,q}}(v)$ constructed by applying the six steps of procedure *IntroduceNode* on $C_u \in \mathrm{FSC}_{\mathrm{k,q}}(u)$.

We assume that C_v is obtained from C_u with Case 2 of Step "update T_u into T_v" of Procedure *IntroduceNode*. That is, C_v is obtained from C_u by adding v_{leaf} as a neighbor of v_{att}, where v_{att} result from the subdivision of $f = \{v_{\mathit{top}}, v_{\mathit{bottom}}\} \in E(T_u)$. Case 1 of Step 1 can be proved in a similar way, thus we omit the proof here.

By definition $C_u \in \mathrm{FSC}_{\mathrm{k,q}}(u)$, hence it is a characteristic of a partitioning tree $(T_u^\circ, r_u^\circ, \sigma_u^\circ)$ of A_u restricted to X_u, i.e. $C_u = \mathrm{Char}((T_u^\circ, r_u^\circ, \sigma_u^\circ), X_u)$. Since $\mathrm{FSC}_{\mathrm{k,q}}(u)$ is a full set of (k,q)-characteristics, we have that $(T_u^\circ, r_u^\circ, \sigma_u^\circ)$ is a q-branched partitioning tree for A_u with Φ-width at most k.

Note that, by definition of $\mathrm{Char}((T_u^\circ, r_u^\circ, \sigma_u^\circ), X_u)$ and the "Contr" operation, f represents an edge $f_u^\circ \in E(T_u^\circ)$.

Let $(T_v^\circ, r_v^\circ, \sigma_v^\circ)$ be obtained from $(T_u^\circ, r_u^\circ, \sigma_u^\circ)$ by subdividing the edge $f_u^\circ = \{v_{\mathit{top}}^\circ, v_{\mathit{bottom}}^\circ\}$ one time, creating a vertex v_{att}, and adding v_{leaf} as neighbor of v_{att}, make $\sigma_v^\circ(v_{\mathit{leaf}}) = a$ and $r_v^\circ = r_u^\circ$. By definition, $(T_v^\circ, r_v^\circ, \sigma_v^\circ)$ is a partitioning tree of A_v.

Let $C_s = ((T_s, r_s, \sigma_s), \ell_s, K_s, \mathrm{dist}_s, \mathit{out}_s, \mathit{branch}_s, \mathit{father}_s) = \mathrm{Char}((T_v^\circ, r_v^\circ, \sigma_v^\circ), X_v)$, i.e. C_s is the restriction of $(T_v^\circ, r_v^\circ, \sigma_v^\circ)$ to X_v.

We need to show that $C_v = C_s$.

In order to prove that $C_v = C_s$, we need to introduce some notation.

For all $t \in V(\mathrm{cp}(T_v^\circ))$, let $\mathcal{A}_t^\circ$ be the partition of A_v defined by t. Similarly, for all $e \in E(T_v^\circ)$, let $\mathcal{A}_e^\circ$ be the partition of A_v defined by e.

Let T'_v be the smallest subtree of T_v° the leaves of which map all elements of X_v, and let r'_v be the vertex in T'_v that is closest to r_v°. Similarly, let T'_u be the smallest subtree of T_u° the leaves of which map all elements of X_u, and let r'_u be the vertex in T'_u that is closest to r_u°. Note that, T'_u and T'_v are obtained in the first step of the procedure Char when applied to $(T_u^\circ, r_u^\circ, \sigma_u^\circ)$ and $(T_v^\circ, r_v^\circ, \sigma_v^\circ)$ respectively.

For the remainder of this section, assume that $V(T_v^\circ) = \{1, \ldots, n\}$, where $v_{leaf} = n$ and $v_{att} = n-1$. Then, from the definition of T_v°, $V(T_u^\circ) = V(T_v^\circ) \setminus \{n-1, n\}$. Moreover, from the Char procedure, $V(T_s) \subseteq V(T_v^\circ)$ and $V(T_u) \subseteq V(T_u^\circ)$ and, from the *IntroduceNode* procedure $V(T_v) \subseteq V(T_u) \cup \{v_{leaf}, v_{att}\}$.

Claim 2. *The sets K_v and K_s are the same, i.e. $K_v = K_s$.*

By step 3 of the procedure Char, K_s is the set of vertices that are leaves in (T'_v, r'_v), parents of leaves, branching nodes of $\mathrm{cp}(T_v^\circ)$ in $V(T'_v)$, or branching nodes of (T'_v, r'_v). T_v° can be obtained from T_u° by subdividing f_u° and adding a new leaf v_{leaf} adjacent to the new vertex v_{att} and $\sigma_v^\circ(v_{leaf}) = a$, therefore T'_v is obtained from T'_u by subdividing f_u° and adding a neighbor to the vertex created from the subdivision.

$K_s \setminus \{v_{top}\}$ is composed by vertices that are leaves in (T'_u, r'_u), or parents of leaves, or branching nodes of $\mathrm{cp}(T_u^\circ)$ in $V(T'_u)$, or branching nodes of (T'_u, r'_u), or v_{att}, or v_{leaf}. In other words, $K_s \setminus \{v_{top}\} = (K_u \cup \{v_{att}, v_{leaf}\}) \setminus \{v_{top}\}$.

There are 2 cases to consider: (1) v_{bottom} is the unique child of v_{top} in T_u and $father_u(v_{top}) = 0$; or (2) otherwise.

Case (1): $K_v = (K_u \cup \{v_{att}, v_{leaf}\}) \setminus \{v_{top}\}$. Since v_{bottom} is the unique child of v_{top} in T_u and $father_u(v_{top}) = 0$, v_{top} has only one child in T'_u and it is not a branching node of T_u°. From the construction of T_v°, the only child of v_{top} in T'_v is v_{att} which is not a leaf and v_{top} is not a branching node of T_v°. Hence, $v_{top} \notin K_s$. Therefore, $K_s \setminus \{v_{top}\} = K_s = K_v$.

Case (2): either v_{top} has more than one child in T_u or $father_u(v_{top}) = 1$. Then, $K_v = K_u \cup \{v_{att}, v_{leaf}\}$, from step "creation of a new branching node" of *IntroduceNode*. There are some sub-cases to consider:

- If v_{top} has more than one child in T_u, then v_{top} is a branching node of (T'_u, r'_u), hence $v_{top} \in K_u$. From the construction of T'_v, v_{top} is a branching node of (T'_v, r'_v), therefore $v_{top} \in K_s$. Then, since $v_{top} \in K_u$ and $K_s \setminus \{v_{top}\} = (K_u \cup \{v_{att}, v_{leaf}\}) \setminus \{v_{top}\}$ we have that $K_s = K_u \cup \{v_{att}, v_{leaf}\} = K_v$.
- If $father_u(v_{top}) = 1$ and v_{bottom} is a leaf of T_u, then $v_{top}^\circ \in V(T_u^\circ)$ is the parent of a leaf of T_u°, hence $v_{top} \in K_u$. From the construction of T_v°, v_{top} is a branching node of $\mathrm{cp}(T_v^\circ)$, hence $v_{top} \in K_s$. Then, $K_s = K_u \cup \{v_{att}, v_{leaf}\} = K_v$.
- If $father_u(v_{top}) = 1$ and v_{bottom} is not a leaf of T_u. From the construction of T_v°, v_{top} is a branching node of T_v° if and only if v_{top} is a branching node of T_u°. Hence, $v_{top} \in K_u$ if and only if $v_{top} \in K_s$. Therefore, $K_s = K_u \cup \{v_{att}, v_{leaf}\} = K_v$.

In both cases, we have $K_v = K_s$.

Claim 3. *T_v and T_s are isomorphic and the labels of correspondent vertices of T_v and T_s are the same. That is, $\ell_v(t_v) = \ell_s(t_s)$ for all internal vertex or edge t_v of T_v that has a corresponding vertex or edge t_s in T_s.*

Recall that, from the Char procedure, T_s is obtained by contracting all paths of T_v° that have endpoints in K_s and no internal vertex in K_s. On the other hand, T_v is obtained by adding a v_{leaf} adjacent to v_{att} in T_u and then contracting all paths of T_v that have endpoints in K_v and no internal vertex in K_v. We want to show that the result of the contractions in T_v° and the contractions in T_v are the same. That is, that after contractions T_v and T_s are isomorphic having the same labels on its vertices.

Let $P_v(x,y)$ denote the labeled path between vertices x and y in T_v with labels ℓ_v (resp. $P_s(x,y)$ in T_s with labels ℓ_s). In order to show these two properties, we show that for each pair of vertices $x, y \in K_v = K_s$ (Claim 2) such that the path $P_v(x,y)$ has no vertex from K_v as internal vertex then $P_v(x,y) = P_s(x,y)$. In other words, the paths in T_v and T_s between two vertices of $K_v = K_s$ have the same length and have the same sequence of labels defined by the functions ℓ_v and ℓ_s respectively.

Let $P'_s(x,y)$ be the path between x and y in T'_v. In other words, $P'_s(x,y)$ is the path $P_s(x,y)$ of C_s along with labels given by ℓ_s before Step 6 of Char$((T_v^\circ, r_v^\circ, \sigma_v^\circ), X_v)$.

There are some cases to consider: (1) $P_s(x,y) = (v_{att}, v_{leaf})$; (2) $P_s(x,y) \neq (v_{att}, v_{leaf})$ and $v_{att} \notin \{x,y\}$; (3) $P_s(x,y) \neq (v_{att}, v_{leaf})$ and $v_{att} \in \{x,y\}$.

Case (1): If $P_s(x,y) = \{v_{att}, v_{leaf}\}$, by Step "update labels of new vertex(s) and edge(s)" of Procedure *IntroduceNode* $\ell_v(\{v_{att}, v_{leaf}\}) = \ell_s(\{v_{att}, v_{leaf}\})$. From the fact that C_s is obtained through the Char procedure, we have that $\ell_s(v_{att}) = \Phi_{A_v}(\mathcal{A}_v^\circ)$. From the fact that Φ is compatible with $(D, \mathcal{X})$, we have that $\Phi_{A_v}(\mathcal{A}_v^\circ) = F_\Phi(\ell_u(f), \mathcal{A}_v^\circ \cap X_v, a)$. Finally, from the step "update of labels of vertices and edges" of procedure *IntroduceNode* we have that $\ell_v(v_{att}) = F_\Phi(\ell_u(f), \mathcal{A}_v^\circ \cap X_v, a)$. Taking all these inequalities we have that $\ell_s(v_{att}) = \Phi_{A_v}(\mathcal{A}_v^\circ) = F_\Phi(\ell_u(f), \mathcal{A}_v^\circ \cap X_v, a) = \ell_v(v_{att})$.

Case (2): Now, let us assume that $P_s(x,y) \neq \{v_{att}, v_{leaf}\}$ and $v_{att} \notin \{x,y\}$. Then, $P_s(x,y)$ represents a path P_u° in T_u°, and more precisely in T'_u. Each $t \in V(P_u^\circ)$ defines a partition $\mathcal{T}_t^\circ$ of A_u such that $\Phi_{A_u}(\mathcal{T}_t^\circ) = \Phi_{A_v}(\mathcal{A}_t^\circ) \cap A_u$. Similarly, each $e \in E(P_u^\circ)$ defines a partition $\mathcal{T}_e^\circ$ of A_u such that $\Phi_{A_u}(\mathcal{T}_e^\circ) = \Phi_{A_v}(\mathcal{A}_e^\circ) \cap A_u$. Moreover, the labels in P_u° are given by the function Φ_{A_u} applied to the partitions of A_u defined by the vertices (or edges) of P_u°.

When computing Char$((T_u^\circ, r_u^\circ, \sigma_u^\circ), X_u)$ to obtain C_u, P_u° is replaced by Contr(P_u°). Each internal vertex and edge of Contr(P_u°) defines the same partition $\mathcal{P}$ of X_v (where a is in the part correspondent to the component where edge f is), since these vertices are not in K_u, hence they are not leaves, nor parent of leaves of X_u, nor branching nodes of cp(T_u°) and f_u° does not belong to P_u°.

To obtain $P_v(x,y)$, procedure *IntroduceNode* modifies the labels of edges and vertices of Contr(P_u°) by applying the strictly increasing function $F_{\Phi,\mathcal{P}} : x \to F_\Phi(x, \mathcal{P}, a)$ (Step "update of labels of vertex(s) and edge(s)" of Procedure *IntroduceNode*), then, let $F_{\Phi,\mathcal{P}}$(Contr(P_u°)) be the path obtained in this way, then it replaces $F_{\Phi,\mathcal{P}}$(Contr(P_u°)) by Contr$(F_{\Phi,\mathcal{P}}$(Contr$(P_u^\circ)))$. Hence, Contr$(F_{\Phi,\mathcal{P}}$(Contr$(P_u^\circ))) = P_v(x,y)$. By Items 2 and 1 of Lemma 23,

$$P_v(x,y) = \text{Contr}(F_{\Phi,\mathcal{P}}(\text{Contr}(P_u^\circ))) = \text{Contr}(\text{Contr}(F_{\Phi,\mathcal{P}}(P_u^\circ))) = \text{Contr}(F_{\Phi,\mathcal{P}}(P_u^\circ)).$$

From the definition of T_v° and the fact that Φ is compatible with $(D, \mathcal{X})$, we have that $F_{\Phi,\mathcal{P}}(P_u^\circ) = P_v^\circ$, hence, by Step 6 of Char$((T_v^\circ, r_v^\circ, \sigma_v^\circ), X_v)$, Contr$(F_{\Phi,\mathcal{P}}(P_u^\circ)) = P_s(x,y)$. Therefore, $P_v(x,y) = P_s(x,y)$.

Case (3): Let x and y be the vertices in K_u such that the path P_u° of T_u° between x and y contains the edge f_u°. It remains to prove that $P_v(x, v_{att}) = P_s(x, v_{att})$ and $P_v(v_{att}, y) = P_s(v_{att}, y)$.

Let C'_u be obtained from C_u by subdividing f resulting in new vertex v_{att}. Moreover, let $\ell'_u(v_{att}) = \ell'_u(\{v_{top}, v_{att}\}) = \ell'_u(\{v_{att}, v_{bottom}\}) = \ell_u(f)$. From the procedure *IntroduceNode* $P_v(x, v_{att})$ is obtained from $P'_u(x, v_{att})$ by applying the function F and then a contraction and $P_v(v_{att}, y)$ is obtained from $P'_u(v_{att}, y)$ by applying the function F and then a contraction. That is, $P_v(x, v_{att}) = \text{Contr}(F_{\Phi,\mathcal{P}}(P'_u(x, v_{att})))$, where $\mathcal{P}$ is the partition of $\mathcal{P}$ is the partition of X_v defined by vertices and edges in $P_v(x, v_{att})$. Similarly, $P_v(v_{att}, y) = \text{Contr}(F_{\Phi,\mathcal{P}'}(P'_u(v_{att}, y)))$, where $\mathcal{P}'$ is the partition of X_v defined by vertices and edges in $P_v(v_{att}, y)$.

On the other hand, $P_s(x, v_{att})$ is obtained by applying a contraction on the path from x to v_{att} in T_v° and $P_s(v_{att}, y)$ is obtained by applying a contraction on the path from v_{att} to y.

Let T''_u be the tree obtained from T_u° by subdividing f_u° resulting in new vertex v_{att}, i.e. T''_u is the tree T_v° without v_{leaf}. Let $P''_u(x, y) = P''_u(x, v_{att}) \odot P''_u(v_{att}, y)$ be the path in T''_u between x and y. Then, $P_v^\circ(x, y) = P_v^\circ(x, v_{att}) \odot P_v^\circ(v_{att}, y)$ is the path in T_v° corresponding to $P''_u(x, y)$ in T''_u. That is, $P_v^\circ(x, y)$ has the same vertices as $P''_u(x, y)$ but with different labels.

Note that the partition of X_v defined by the vertices and edges in $P_v^\circ(x, v_{att})$ is the same as the one defined by $P_v(x, v_{att})$, that is, $\mathcal{P}$. Similarly, the partition of X_v defined by the vertices and edges in $P_v^\circ(v_{att}, y)$ is the same as the one defined by $P_v(v_{att}, y)$, that is, $\mathcal{P}'$. From the fact that the fact Φ is compatible with $(D, \mathcal{X})$, we have that $P_v^\circ(x, v_{att}) = F_{\Phi,\mathcal{P}}(P''_u(x, v_{att}))$ and that $P_v^\circ(v_{att}, y) = F_{\Phi,\mathcal{P}'}(P''_u(v_{att}, y))$. Therefore, $P_s(x, v_{att}) = \text{Contr}(P_v^\circ(x, v_{att})) = \text{Contr}(F_{\Phi,\mathcal{P}}(P''_u(x, v_{att})))$ and $P_s(v_{att}, y) = \text{Contr}(P_v^\circ(v_{att}, y)) = \text{Contr}(F_{\Phi,\mathcal{P}'}(P''_u(v_{att}, y)))$.

Then, taking all these inequalities, we have that:

$$\begin{aligned}
P_v(x, v_{att}) &= \text{Contr}(F_{\Phi,\mathcal{P}}(P'_u(x, v_{att}))),\\
P_v(v_{att}, y) &= \text{Contr}(F_{\Phi,\mathcal{P}'}(P'_u(v_{att}, y))),\\
P_s(x, v_{att}) &= \text{Contr}(F_{\Phi,\mathcal{P}}(P''_u(x, v_{att}))) \text{ and}\\
P_s(v_{att}, y) &= \text{Contr}(F_{\Phi,\mathcal{P}'}(P''_u(v_{att}, y))).
\end{aligned}$$

Note that $P'_u(x, y) = \text{Contr}(P''_u(x, y))$, then by Lemma 24:

$$\begin{aligned}
P_s(x, v_{att}) &= \text{Contr}(F_{\Phi,\mathcal{P}}(P''_u(x, v_{att}))) = \text{Contr}(F_{\Phi,\mathcal{P}}(P'_u(x, v_{att}))) = P_v(x, v_{att}), \text{ and}\\
P_s(v_{att}, y) &= \text{Contr}(F_{\Phi,\mathcal{P}'}(P''_u(v_{att}, y))) = \text{Contr}(F_{\Phi,\mathcal{P}'}(P'_u(v_{att}, y))) = P_v(v_{att}, y).
\end{aligned}$$

Since $K_v = K_s$ (Claim 2, and, in all cases, for each x, $y \in K_s$, $P_v(x, y) = P_s(x, y)$, we have that T_v is isomorphic to T_s and that ℓ_v and ℓ_s are equivalent, that is, correspondent vertices and edges in T_v and T_s have the same label.

Claim 4. $(\text{dist}_v, out_v, branch_v, father_v) = (\text{dist}_s, out_s, branch_s, father_s)$.

By procedure *IntroduceNode*, dist_v receives the value of dist_u, i.e. the number of branching nodes in T_u° between r_u° and r'_u. We have that dist_u is the number of branching nodes in T_v° between r_v° and r'_s, i.e. dist_s. Hence, $\text{dist}_v = \text{dist}_s$.

Now, for every vertex t in $\mathrm{cp}(T_v)$, $out_v(t)$ is the maximum number of branching nodes on a path between t and a leaf in $A_u \setminus X_u$ every internal vertices of which are different from r_u° and in $T'_u \setminus T_u^\circ$. It is also the maximum number of branching nodes on a path between t and a leaf in $A_v \setminus X_v = A_u \setminus X_u$ every internal vertices of which are different from r_v° and in $T'_s \setminus T_v^\circ$, i.e. $out_s(t)$.

For every vertex t in $\mathrm{cp}(T_v)$, $branch_v(t) = 1$ if and only if $branch_u(t) = 1$ or t is the parent-end of f and $father_u(t) = 1$ (Step "creation of a new branching node" of Procedure *IntroduceNode*). That is, $branch_v(t) = 1$ if and only if t is a branching node of $\mathrm{cp}(T_v^\circ)$, i.e. $branch_v(t) = branch_s(t)$.

Now, for every vertex t in $\mathrm{cp}(T_v)\setminus\{v_{top}\}$, $father_v(t) = father_u(t)$ and $father_v(v_{top}) = 1$. In other words, $father_v(t) = 1$ if and only if t is the parent of a non leaf node in T_v°. Since, $father_u(t) = father_s(t)$ for every vertex $t \in \mathrm{cp}(T_u)$, $father_v(t) = father_s(t)$. Moreover, $father_s(v_{top}) = 1$.

Claim 5. *$(T_v^\circ, r_v^\circ, \sigma_v^\circ)$ is a q-branched partitioning tree for A_v with Φ-width not bigger than k.*

Therefore, we proved that $C_v = \mathrm{Char}((T_v^\circ, r_v^\circ, \sigma_v^\circ), X_v)$. By Step "update of $\mathrm{FSC_{k,q}}(v)$" of Procedure *IntroduceNode*, we have that $\mathrm{dist}_v + \mathrm{brheight}_T(r_v) \leq q$. Note that, since (T_u°, r_u°) is q-branched, for every path P in $\mathrm{cp}(T_v^\circ)$ from r_v° to a leaf of $\mathrm{cp}(T_v^\circ)$ such that P does not pass through r'_v we have that P has at most q branching nodes. Hence, by Lemma 26 $(T_v^\circ, r_v^\circ, \sigma_v^\circ)$ is q-branched.

It remains to show that $(T_v^\circ, r_v^\circ, \sigma_v^\circ)$ has Φ width at most k. Consider any internal vertex t of $V(T_v^\circ) \setminus V(T'_v)$. Let $\mathcal{P}_t$ be the partition of A_v defined by t. Since t is not in $V(T'_v)$ the partition of X_v it defines has only one part. That is, the partition $\mathcal{P}_t \cap X_v$ defined by t has at most one part. From the fact that Φ is compatible with $(D, \mathcal{X})$ we have that $\Phi_{A_v}(\mathcal{P}_t) = \Phi_{A_v}(\mathcal{P}_t) = \Phi_{A_u}(\mathcal{P}_t \cap A_u)$. Then, from the fact that $(T_v^\circ, r_v^\circ, \sigma_v^\circ)$ has Φ width at most k we have that $\Phi_{A_v}(\mathcal{P}_t) \leq k$.

Similarly, for any edge e of $E(T_v^\circ) \setminus E(T'_v)$ we have that $\Phi_{A_v}(\mathcal{P}_e) \leq k$, where $\mathcal{P}_e$ is the partition of A_v defined by e.

From the definition of "Contr", item 1 from Lemma 22 and the fact that $\ell_v(t) \leq k$ for any vertex $t \in V(\mathrm{cp}(T_v))$, and $\ell_v(e) \leq k$ for any edge $e \in E(T_v)$, we have that (T'_v, r'_v, σ'_v) has Φ-width at most k. Hence, $(T_v^\circ, r_v^\circ, \sigma_v^\circ)$ has Φ-width at most k.

Therefore, $(T_v^\circ, r_v^\circ, \sigma_v^\circ)$ is a q-branched partitioning tree with Φ-width at most k. Thus, $C_v = \mathrm{Char}((T_v^\circ, r_v^\circ, \sigma_v^\circ), X_v)$ is a (k, q)-characteristic of $(T_v^\circ, r_v^\circ, \sigma_v^\circ)$ restricted to X_v.

This concludes the proof, that is $\mathrm{FSC_{k,q}}(v)$ is a set of (k, q)-characteristics restricted to A_v. □

To prove that the set $\mathrm{FSC_{k,q}}(v)$ is full, we consider an arbitrary q-branched partitioning tree $(T_v^\circ, r_v^\circ, \sigma_v^\circ)$ with Φ-width not bigger than k of A_v and show that there is an execution of *IntroduceNode* on a characteristic $C_u \in \mathrm{FSC_{k,q}}(u)$ such that $C_v = \mathrm{Char}((T_v^\circ, r_v^\circ, \sigma_v^\circ), X_v)$ and $C_v \in \mathrm{FSC_{k,q}}(v)$.

Lemma 33. *The set $\mathrm{FSC_{k,q}}(v)$, computed through the procedure* IntroduceNode*, is full.*

Proof. Let $(T_v^\circ, r_v^\circ, \sigma_v^\circ)$ be a q-branched partitioning tree of A_v with Φ-width not bigger than k. Let v_{leaf} be the leaf of T_v° that maps $\{a\} = X_v \setminus X_u$. Let v_{att} be the parent of v_{leaf}. Let $(T_u^\circ, r_u^\circ, \sigma_u^\circ)$ be the partitioning tree of A_u such that $(T_v^\circ, r_v^\circ, \sigma_v^\circ)$ can be obtained from $(T_u^\circ, r_u^\circ, \sigma_u^\circ)$ by inserting a vertex correspondent to a as a neighbor of v_{att} in T_u°. Therefore, $(T_u^\circ, r_u^\circ, \sigma_u^\circ)$ is a q-branched partitioning tree for A_u with Φ-width not bigger than k. Let C_v be such that $C_v = \mathrm{Char}((T_v^\circ, r_v^\circ, \sigma_v^\circ), X_v)$. It remains to show that $C_v \in \mathrm{FSC_{k,q}}(v)$.

From the induction hypothesis, there is $C_u = \text{Char}((T_u^\circ, r_u^\circ, \sigma_u^\circ), X_u)$ in $\text{FSC}_{\text{k,q}}(u)$. Then, there are three cases to consider: (1) $v_{att} \in K_u$; (2) $v_{att} \notin K_u$ and v_{att} is represented in C_u; (3) $v_{att} \notin K_u$ and v_{att} is not represented in C_u.

Cases (1) and (2): Then, v_{att} is a vertex of T_u, i.e. during the Step 6 of $\text{Char}((T_u^\circ, r_u^\circ, \sigma_u^\circ), X_u)$ the vertex v_{att} does not suffer a contraction operation. Consider the execution of case 1 of procedure *IntroduceNode* on C_u, where v_{leaf} is added as a neighbor to v_{att}, to obtain C_v. In other words, T_v can be obtained from T_u by adding a vertex v_{leaf} as a neighbor of v_{att}. Hence, from the step "update of labels of vertices and edges" of *IntroduceNode* procedure and the fact that Φ is compatible with $(D, \mathcal{X})$, the labels ℓ_v of C_v obtained from the labels ℓ_u of C_u are such that for any internal vertex (or edge) t in T_v:

$$\ell_v(t) = F_\Phi(\ell_u(t), \mathcal{T}_t, a) = F_\Phi(\Phi_{A_u}(\mathcal{A}_t \cap A_u), \mathcal{A}_t \cap X_v, a) = \Phi_{A_v}(\mathcal{A}_t)$$

Where $\mathcal{T}_t$ and $\mathcal{A}_t$ are the partitions of X_v and A_v defined by t respectively. From the Step "update of labels of new vertex(s) and edge(s)" of *IntroduceNode* procedure $\ell_v(\{v_{att}, v_{leaf}\}) = \Phi_{A_v}(\{A_u, \{a\}\})$.

We need to show that in step "update of $\text{FSC}_{\text{k,q}}(v)$" we add C_v to $\text{FSC}_{\text{k,q}}(v)$. Since $(T_v^\circ, r_v^\circ, \sigma_v^\circ)$ is q-branched and its Φ-width is not bigger than k and $C_v = \text{Char}((T_v^\circ, r_v^\circ, \sigma_v^\circ), X_v)$, we have that $\text{brheight}(r_v) \leq q$ and that, for all internal vertices (or edges) t of T_v, $\ell_v(t) \leq k$ (Lemma 27). Hence, during step "update of $\text{FSC}_{\text{k,q}}(v)$" we have that C_v is inserted into $\text{FSC}_{\text{k,q}}(v)$.

Case (3): The vertex v_{att} does not belong to K_u nor has a representative on C_u. This means that v_{att} is contracted during the operation $\text{Char}((T_u^\circ, r_u^\circ, \sigma_u^\circ))$. Let $P'_u(x, y)$ be the path between x and y in T'_u (the subtree of T_u° spanning leaves mapping elements of X_u) such that $v_{att} \in V(P'_u(x, y))$ and $x, y \in K_u$. In other words, in Step 6 of $\text{Char}((T_u^\circ, r_u^\circ, \sigma_u^\circ), X_u)$ to obtain C_u during the contraction of $P'_u(x, y)$ we have that v_{att} is removed with a contraction operation. Let $P_u(x, y) = \text{Contr}(P'_u(x, y))$, i.e. the path in C_u resulting from the contraction of $P'_u(x, y)$. Thus, there are vertices x' and y' in $V(P_u(x, y))$ such that v_{att} is an internal node of $P'_u(x', y')$ and $\{x', y'\}$ is an edge of $P_u(x, y)$. In other words, v_{att} is removed either by a contraction between $\{x', y'\}$ and x' or by a contraction between $\{x', y'\}$ and y'.

We want to show that C_v, obtained through an execution of case 2 of *IntroduceNode* on C_u where the edge $\{x', y'\}$ is subdivided, is such that $C_v = \text{Char}((T_v^\circ, r_v^\circ, \sigma_v^\circ), X_v)$ and $C_v \in \text{FSC}_{\text{k,q}}(v)$.

Consider the execution of procedure *IntroduceNode* on C_u by applying case 2 on the edge $\{x', y'\}$. In this execution of *IntroduceNode*, T_v is obtained from T_u by subdividing $\{x', y'\}$ creating a vertex v_{att} and adding v_{leaf}, mapping a, as a neighbor of v_{att}.

Using the same argument as in "proof of cases (1) and (2)" we have that for any internal vertex (or edge) t in T_v, $\ell_v(t) = \Phi_{A_v}(\mathcal{A}_t)$ where $\mathcal{A}_t$ is the partition of A_v defined by t.

We need to show that in step "update of $\text{FSC}_{\text{k,q}}(v)$" we add C_v to $\text{FSC}_{\text{k,q}}(v)$. Since $(T_v^\circ, r_v^\circ, \sigma_v^\circ)$ is q-branched and its Φ-width is not bigger than k and $C_v = \text{Char}((T_v^\circ, r_v^\circ, \sigma_v^\circ), X_v)$, we have that $\text{brheight}(r_v) \leq q$ and that, for all internal vertices (or edges) t of T_v, $\ell_v(t) \leq k$ (Lemma 27). Hence, during step "update of $\text{FSC}_{\text{k,q}}(v)$" we have that C_v is inserted into $\text{FSC}_{\text{k,q}}(v)$.

Since, in all these cases, we have that $C_v \in \mathrm{FSC}_{\mathrm{k,q}}(v)$, this shows that $\mathrm{FSC}_{\mathrm{k,q}}(v)$ is a full set of (k,q)-characteristics for A_v restricted to X_v. □

Theorem 34. *Procedure* IntroduceNode *computes a full set of (k,q)-characteristics of A_v restricted to X_v in time that does not depend on $|A|$. That is the complexity of procedure* IntroduceNode *is bounded by a a function $f_i(k,q,k') = O((60kqk')^{45kqk'}(kqk')^4)$, if $q < \infty$, or a function $f_i'(k,k') = O((15kk')^{45kk'}(kk')^4)$ otherwise.*

Proof. From Theorem 31, procedure *IntroduceNode* computes a full set of (k,q)-characteristics of A_v restricted to X_v. It remains to prove that this can be done time that does not depend on $|A|$.

Assume that $q < \infty$, the case where $q = \infty$ is similar and thus omitted. From its definition, F_Φ can be computed in constant time. Therefore, for each element $C_u = ((T_u, r_u, \sigma_u), \ell_u, K_u, \mathrm{dist}_u, out_u, branch_u, father_u) \in \mathrm{FSC}_{\mathrm{k,q}}(u)$ steps 1 to 5 can be done in $O(|T_u|)$, for each possible execution of Step 1.

Contracting a path P can be done in $O(|P|^3)$, by taking all possible pairs of vertices and edges and verifying if a contraction operation can be done between them. Hence, step 5 can be executed in $O(|T_u|^3)$, for each possible execution of Step 1.

Lastly, step 6 can be executed in $O(|T_u|)$, by traversing the tree T_u in a bottom up order, for each possible execution of Step 1.

Since the size and the number of elements in $\mathrm{FSC}_{\mathrm{k,q}}(u)$ is bounded by $f'(k,q,k') = O(kqk')$ and $f(k,q,k') = O((60kqk')^{45kqk'})$ respectively, if $q < \infty$, or by $f'(k,k') = O(kk')$ and $f(k,k') = O((15kk')^{45kk'})$ if $q = \infty$ from Lemma 28, we get the result. That is, since there are at most $O(|T_u|)$ different executions for Step 1, the complexity of procedure *IntroduceNode* is bounded by, if $q < \infty$:

$$f_i(k,q,k') = O\left(f(k,q,k') \cdot f'(k,q,k')^4\right) = O((60kqk')^{45kqk'}(kqk')^4).$$

In the case that $q = \infty$ the complexity of procedure *IntroduceNode* is bounded by:

$$f_i(k,k') = O\left(f(k,k') \cdot f'(k,k')^4\right) = O((15kk')^{45kk'}(kk')^4).$$

□

Procedure *ForgetNode*

Let v be a forget node of D, u be its child and $\mathrm{FSC}_{\mathrm{k,q}}(u)$ be a full set of (k,q)-characteristics of A_u restricted to X_u. For every characteristic $C_u = ((T_u, r_u, \sigma_u), \ell_u, K_u, \mathrm{dist}_u, out_u, branch_u, father_u) \in \mathrm{FSC}_{\mathrm{k,q}}(u)$, Procedure *ForgetNode* proceeds as follows. Roughly, it restricts C_u to $X_v = X_u \setminus \{a\}$ obtaining C_v.

1. *preparation:*

   Let v_{leaf} be the leaf of T_u that maps a, let v_{att} be the vertex of T_u with degree at least three that is closest to v_{leaf} (if no such a vertex exists, T_u is a path and v_{att} is set to the only other leaf of the path). Let P be the path between v_{leaf} and v_{att}. Let w be the neighbor of v_{att} in P. Let p be the number of vertices $y \in V(T_u) \setminus \{v_{att}\}$ with $branch_u(y) = 1$ in the path between v_{att} and r_u.

2. *removing a from T_u:*

   T_v is obtained by removing $V(P) \setminus \{v_{att}\}$ and $E(P)$ from T_u.

   If $r_u \neq v_{att}$ and r_u belongs to the path P between v_{leaf} and v_{att}:

- $r_v \leftarrow v_{att}$,
- $\mathrm{dist}_v \leftarrow \mathrm{dist}_u + p$, and
- $out_v(v_{att}) \leftarrow out_u(v_{att})$.

If $r_u = v_{att}$ or r_u does not belong to P:

- $r_v \leftarrow r_u$,
- $\mathrm{dist}_v \leftarrow \mathrm{dist}_u$, and
- $out_v(v_{att}) \leftarrow \max\{out_u(v_{att}), \mathrm{brheight}_T(w)\}$.

In either case:

- $branch_v(v_{att}) \leftarrow branch_u(v_{att})$, and
- $father_v(v_{att}) \leftarrow father_u(v_{att})$.

For every vertex $x \in \mathrm{cp}(T_v)$ such that $x \neq v_{att}$:

- $out_v(x) \leftarrow out_u(x)$,
- $branch_v(x) \leftarrow branch_u(x)$, and
- $father_v(x) \leftarrow father_u(x)$.

3. *updating K_v:*

   If v_{att} has degree two in T_v, and v_{att} is not the parent of a leaf neither the root in T_v, and $branch_u(v_{att}) = 0$, then $K_v \leftarrow K_u \setminus V(P)$.

   Otherwise, K_v is obtained by removing $V(P) \setminus \{v_{att}\}$ from K_u.

4. *contracting the paths:*

   $\forall x, y \in K_v$ and path P between x and y such that no internal vertices of P are in K_v, $P \leftarrow \mathrm{Contr}(P)$.

5. *updating* $\mathrm{FSC}_{\mathrm{k,q}}(v)$*:*

   Add C_v to $\mathrm{FSC}_{\mathrm{k,q}}(v)$.

The rest of this section is dedicated to proving Lemma 35.

Since $A_v = A_u$, A_v admits a q-branched partitioning tree with Φ-width at most k only if A_u does. Therefore, we can assume that $\mathrm{FSC}_{\mathrm{k,q}}(u) \neq \emptyset$, otherwise, A_v does not admit a q-branched partitioning tree with Φ-width at most k, and $\mathrm{FSC}_{\mathrm{k,q}}(v) = \emptyset$. A scheme of the proof of Lemma 35 can be found in Figure 4.7

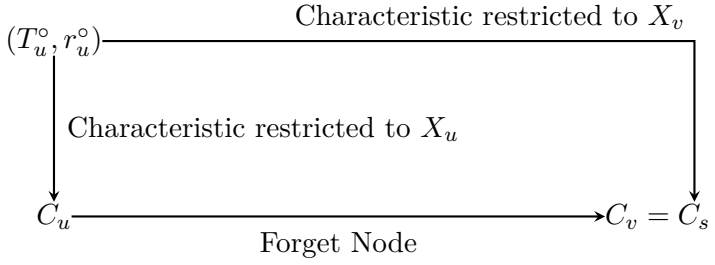


Figure 4.7: Scheme of proof of Lemma 35.

Lemma 35. ForgetNode *computes a full set of* (k,q)*-characteristics of* A_v *restricted to* X_v.

Proof. Let $(T^\circ, r^\circ, \sigma^\circ)$ be any q-branched partitioning tree for A_u with Φ-width at most k. Since $A_v = A_u$, we have that $(T^\circ, r^\circ, \sigma^\circ)$ is a q-branched partitioning tree for A_u with Φ-width at most k if an only if it is also a q-branched partitioning tree for A_v with Φ-width at most k.

Since $\mathrm{FSC}_{\mathrm{k,q}}(u)$ is a full set of (k,q)-characteristics for A_u restricted to X_u, we have that there exists $C_u \in \mathrm{FSC}_{\mathrm{k,q}}(u)$ which is a (k,q)-characteristic of $(T^\circ, r^\circ, \sigma^\circ)$ restricted to X_u. In other words, $C_u = \mathrm{Char}((T^\circ, r^\circ, \sigma^\circ), X_u)$. Let C_v be obtained through procedure *ForgetNode* when applied to C_u, by removing the path $P_u(v_{leaf}, v_{att})$ from T_u, where v_{leaf} is the leaf of T_u mapping a.

We want to show that $C_v = \mathrm{Char}((T^\circ, r^\circ, \sigma^\circ), X_v)$ and that $C_v \in \mathrm{FSC}_{\mathrm{k,q}}(v)$. Since step "updating $\mathrm{FSC}_{\mathrm{k,q}}(v)$" from procedure, we have that $C_v \in \mathrm{FSC}_{\mathrm{k,q}}(v)$. It remains to show that $C_v = \mathrm{Char}((T^\circ, r^\circ, \sigma^\circ), X_v)$.

In order to do that, let $C_s = \mathrm{Char}((T^\circ, r^\circ, \sigma^\circ), X_v)$, we need to show that $C_s = C_v$. Let $C_s = ((T_s, r_s, \sigma_s), \ell_s, K_s, \mathrm{dist}_s, \mathit{out}_s, \mathit{branch}_s, \mathit{father}_s)$ and let $C_v = ((T_v, r_v, \sigma_v), \ell_v, K_v, \mathrm{dist}_v, \mathit{out}_v, \mathit{branch}_v, \mathit{father}_v)$.

To prove that $C_v = C_s$, we must introduce some notation. Let T'_u (resp. T'_v) be the minimum subtree of T° that contains all leaves that maps elements of X_u (resp. X_v).

From the Char procedure, we have that T_u is a path if and only if T'_u is a path. If T'_u is a path, then $|X_u| = 2$. This means that, $|X_v| = 1$ and, hence, T'_v is a single vertex. Then, from the procedure $\mathrm{Char}((T^\circ, r^\circ, \sigma^\circ), X_v)$, we have that C_s is a characteristic of A_v restricted to X_v where T_s is a single vertex. On the other hand, from the step "preparation" of procedure *ForgetNode*, if T_u is a path, we have that T_v is also a single vertex. Hence, it is easy to see that when T'_v is a single vertex $C_v = C_s$.

Therefore, assume that T'_u is not a path and let $v_{att} \in T'_u$ be the vertex with degree at least three that is closest to v_{leaf}, the vertex mapping a.

For the remainder of this section, assume that $V(T^\circ) = \{1, \ldots, n\}$. From the Char procedure, $V(T_s) \subseteq V(T^\circ)$ and $V(T_u) \subseteq V(T^\circ)$ and, from the *ForgetNode* procedure $V(T_v) \subset V(T_u)$.

Claim 6. *The sets* K_v *and* K_s *are the same, i.e.,* $K_s = K_v$.

Let T'_v be the minimum subtree of T° that contains all leaves that maps elements of X_v. Clearly, T'_v is a subtree of T'_u. Let r'_v be the vertex of T'_v that is closest to r°. From the definition of C_s, K_s is the set of vertices of T'_v that are either a leaf of T'_v, or the parent of a leaf of T'_v, or a branching node of $\mathrm{cp}(T^\circ)$ in $V(T'_v)$ (rooted in r°), or a branching node of (T'_v, r'_v).

Since $C_u = \mathrm{Char}((T^\circ, r^\circ, \sigma^\circ), X_u)$, K_u is the set of vertices that are either a leaf of T'_u, or the parent of a leaf of T'_u, or a branching node of $\mathrm{cp}(T^\circ)$ in $V(T'_u)$ (rooted in r°), or a branching node of (T'_u, r'_u). Therefore, since T'_v is a subtree of T'_u and all leaves in T'_v are leaves in T'_u we have that $K_s \subseteq K_u$. Moreover, T'_v is the tree obtained from T'_u by removing the vertex v_{leaf} and all internal vertices of $P(v_{leaf}, v_{att})$, a path between v_{leaf} and v_{att} in T'_u. Hence, $K_s = K_u \setminus V(P(v_{leaf}, v_{att}))$ or $K_s = (K_u \setminus V(P(v_{leaf}, v_{att})) \cup \{v_{att}\})$. There are two cases to consider:

Case (1): Assume that the vertex v_{att} does not belong to K_s. That is, $K_s = K_u \setminus V(P(v_{leaf}, v_{att}))$. Then, v_{att} is not the parent of a leaf in T'_v, nor a branching node of $\mathrm{cp}(T^\circ)$, nor a branching node of (T'_v, r'_v). Since, v_{att} is not the parent of a leaf in

T'_v and it is not a branching node of (T'_v, r'_v) it has degree two in T'_v. Therefore, v_{att} is not the parent of a leaf, different than v_{leaf}, in T'_u, the variable $branch_u(v) = 0$, and v_{att} has degree two in T'_v. Consequently, by Step "updating K_v", $v_{att} \notin K_v$ and $K_v = K_u \setminus V(P_u(v_{leaf}, v_{att}))$, where $P_u(v_{leaf}, v_{att})$ is the path between v_{leaf} and v_{att} in T_u. Hence, $K_s = K_u \setminus V(P(v_{leaf}, v_{att}) = K_u \setminus V(P_u(v_{leaf}, v_{att})) = K_v$.

Case (2): Assume that $v_{att} \in K_s$. That is, $K_s = (K_u \setminus V(P(v_{leaf}, v_{att})) \cup \{v_{att}\})$. If v_{att} belongs to K_s, then it is the parent of a leaf in T'_v, or it is a branching node of $\mathrm{cp}(T^\circ)$, or it is a branching node of (T'_v, r'_v). In all these cases, from the Step "updating K_v", $v_{att} \in K_v$ and $K_v = K_u \setminus V(P_u(v_{leaf}, v_{att})) \cup \{v_{att}\}$, where $P_u(v_{leaf}, v_{att})$ is the path between v_{leaf} and v_{att} in T_u. Hence, $K_s = K_u \setminus V(P(v_{leaf}, v_{att}) \cup \{v_{att}\} = K_u \setminus V(P_u(v_{leaf}, v_{att})) \cup \{v_{att}\} = K_v$.

Hence, $K_s = K_v$.

Claim 7. $r_s = r_v$, $\mathrm{dist}_v = \mathrm{dist}_s$.

From the Char procedure, the root r_s is the vertex of T'_v that is closest to r° in T° and the root r_u is the vertex of T'_u that is closest to r° in T°. Since, T'_v is a subtree of T'_u, r_u is a vertex on the path between r° and r_s.

There are two cases to consider: (1) r_u is not an internal node on the path between v_{leaf} and v_{att}, or (2) r_u is an internal node on the path between v_{leaf} and v_{att}.

Case (1): r_u is not an internal node on the path between v_{leaf} and v_{att}. Then, since T'_v is obtained from T'_u by removing the internal vertices of $P(v_{leaf}, v_{att})$ and the vertex v_{leaf}, we have that $r_u = r_s$. Moreover, since $r_u = r_s$ we have $\mathrm{dist}_s = \mathrm{dist}_u$. Therefore, from Step "removing a from T_u", $r_v = r_u = r_s$ and $\mathrm{dist}_v = \mathrm{dist}_u = \mathrm{dist}_s$.

Case (2): r_u is an internal node on the path between v_{leaf} and v_{att}, then $r_v = v_{att}$. The value dist_u is the number of branching nodes of $\mathrm{cp}(T^\circ)$, excluding the root r°, between r° and r_u.

From Char procedure, $r_s = v_{att}$. The value dist_s is the number of branching nodes of $\mathrm{cp}(T^\circ)$, excluding the root r°, between r° and r_s.

Let $p' = \mathrm{dist}_s - \mathrm{dist}_u$. p' is the number of branching nodes of $\mathrm{cp}(T^\circ)$, excluding the root r°, between r_u and $r_s = v_{att}$. That is, p' is the number of branching nodes of $\mathrm{cp}(T^\circ)$ nodes in the path between r_u and v_{att}. Since, from the Char procedure to obtain C_u, every branching node x of $\mathrm{cp}(T^\circ)$ receives label $branch_u(x) = 1$, we have that p' is the number of nodes x in the path between r_u and v_{att} such that $branch_u(x) = 1$. Therefore, $p' = p$, where p is the value obtained in step "preparation" of procedure *ForgetNode*. Therefore, from Step "removing a from T_u", $\mathrm{dist}_v = p + \mathrm{dist}_u = p' + \mathrm{dist}_u = \mathrm{dist}_s$.

Claim 8. *T_v and T_s are isomorphic and the labels of correspondent vertices of T_v and T_s are the same. That is, $\ell_v(t_v) = \ell_s(t_s)$ for all internal vertex or edge t_v of T_v that has a corresponding vertex or edge t_s in T_s.*

Let $P_v(x, y)$ be a path of C_v such that x and y belongs to K_v and no other internal nodes of $P_v(x, y)$ belongs to K_v. Since $K_v = K_s$, let $P_s(x, y)$ be the corresponding path of C_s between x and y. Assume that the labels of $P_v(x, y)$ and the labels of $P_s(x, y)$ are given by the functions ℓ_v and ℓ_s respectively.

Since $K_v \subseteq K_u$, let $P_u(x, y)$ be the corresponding path of C_u between x and y and let $P^\circ(x, y)$ be the corresponding path in T° between x and y.

We want to show that $P_v(x, y) = P_s(x, y)$. That is, the paths $P_v(x, y)$ and $P_s(x, y)$ have the same sequence of vertices and their labels, given by functions ℓ_v and ℓ_s, are the same. There are two cases to consider, (1) v_{att} is not an internal node of $P_u(x, y)$ or (2) v_{att} is an internal node of $P_u(x, y)$.

Case (1): Assume that v_{att} is not an internal node of $P_u(x, y)$. From the Char procedure, $P_s(x, y) = \text{Contr}(P^\circ(x, y))$ and $P_u(x, y) = \text{Contr}(P^\circ(x, y))$. From the Step "contracting the paths" $P_v(x, y) = \text{Contr}(P_u(x, y))$. Then, from item 1 of Lemma 23 (i.e., for any path P, $\text{Contr}(P) = \text{Contr}(\text{Contr}(P)))$ and $P_v(x, y) = \text{Contr}(P_u(x, y)) = P_s(x, y)$.

Case (2): Assume that v_{att} is an internal node of $P_u(x, y)$. Consequently, $K_v = K_s = K_u \setminus \{v_{att}\}$. From the Char procedure we have that $P_s(x, y) = \text{Contr}(P^\circ(x, y))$ and $P_u(x, y) = \text{Contr}(P^\circ(x, v_{att}) \odot P^\circ(v_{att}, y))$. Then, from Step "contracting the paths" $P_v(x, y) = \text{Contr}(P_u(x, y))$. From item 3 of Lemma 23 (i.e., for any path $P_1 \odot P_2$ we have that $\text{Contr}(P_1 \odot P_2) = \text{Contr}(\text{Contr}(P_1) \odot \text{Contr}(P_2)))$ $P_v(x, y) = P_s(x, y)$.

Therefore, for any pair of vertices x, y in $K_v = K_s$ such that the path between x and y has no vertices in $K_v = K_s$ we have that $P_v(x, y) = P_s(x, y)$. Hence, T_s is isomorphic to T_v and the labels of correspondent vertices of T_v and T_s are the same.

Claim 9. *$\sigma_v = \sigma_s$, $branch_v = branch_s$, $out_v = out_s$ and $father_v = father_s$.*

It is easy to check that $\sigma_v = \sigma_s$, since $T_s = T_v$ and σ_s can be obtained through σ_u by removing the mapping of v_{leaf} to a.

From Step "removing a from T_u" we have that $branch_v = branch_u$. Then, from the induction hypothesis, $branch_u(x) = 1$ if an only if x is a branching node of $\text{cp}(T^\circ)$ in T'_u. Since T'_v is a subtree of T'_u, $branch_u(x) = branch_s(x)$ for all $x \in V(T'_v)$.

Recall that $out_u(x)$ is the maximum number of branching nodes of $\text{cp}(T^\circ)$ in T'_u between x and a leaf of $T^\circ \setminus T'_u$ that does not have any internal node belonging to the set of vertices of T'_u and are not r'_u. Since T'_v is obtained from T'_u by removing the vertices of $V(P(v_{leaf}, v_{att})) \setminus \{v_{att}\}$, for any $x \neq v_{att}$ we have that $out_s(x) = out_u(x)$.

There are two cases to consider to show that $out_v(v_{att}) = out_s(v_{att})$: (1) r'_u is an internal node of $P(v_{att}, v_{leaf})$ in T'_u or (2) r'_u is not an internal node of $P(v_{att}, v_{leaf})$ in T'_u.

Case (1): If r'_u is an internal node of $P(v_{att}, v_{leaf})$, then $out_u(v_{att}) = out_s(v_{att})$. Then, step "removing a from T_u" ensures that $out_v(v_{att}) = out_u(v_{att})$.

Case (2): If r'_u is not an internal node of $P(v_{att}, v_{leaf})$, then $out_s(v_{att})$ is given by $\max\{out_u(v_{att}), \text{brheight}_T\, w\}$ where w is the neighbor of v_{att} in $P(v_{att}, v_{leaf})$. Then, step "removing a from T_u" ensures that $out_v(v_{att}) = \max\{out_u(v_{att}), \text{brheight}(w)\}$.

In both cases, we have $out_v(v_{att}) = out_s(v_{att})$.

For any vertex $x \in T'_u$, $father_s(x) = 1$ if and only if x has a non-leaf child in T°. Hence, $father_s(x) = father_u(x)$. Therefore, from the Step "removing a from T_u" we have $father_s(x) = father_v(x)$.

This proves that $C_v = C_s$. Hence, $C_v = \text{Char}((T^\circ, r^\circ, \sigma^\circ), X_v)$. Moreover, $C_v \in \text{FSC}_{\text{k,q}}(v)$ from step "updating $\text{FSC}_{\text{k,q}}(v)$".

Therefore this proves the lemma. For any q-branched partitioning tree $(T^\circ, r^\circ, \sigma^\circ)$ of A_v with Φ-width at most k we have that $C_v = \mathrm{Char}((T^\circ, r^\circ, \sigma^\circ), X_v) \in \mathrm{FSC}_{\mathrm{k,q}}(V)$.

In other words, $\mathrm{FSC}_{\mathrm{k,q}}(v)$ is a full set of (k,q)-characteristics of A_v restricted to X_v. □

Theorem 36. *Procedure* ForgetNode *computes a full set of (k,q)-characteristics of A_v restricted to X_v in time that does not depend on $|A|$. That is the complexity of procedure* ForgetNode *is bounded by a a function $f_f(k,q,k') = O\left((kqk')^3(60kqk')^{45kqk'}\right)$, if $q < \infty$, or a function $f'_f(k,k') = O\left((kk')^3(15kk')^{45kqk'}\right)$ otherwise.*

Proof. From Theorem 35, procedure *ForgetNode* computes a full set of (k,q)-characteristics of A_v restricted to X_v. It remains to prove that this can be done time that does not depend on $|A|$.

Assume that $q < \infty$, the case where $q = \infty$ is similar and thus omitted.

For each element $C_u = ((T_u, r_u, \sigma_u), \ell_u, K_u, \mathrm{dist}_u, out_u, branch_u, father_u) \in \mathrm{FSC}_{\mathrm{k,q}}(u)$ steps 1 to 3 can be done in $O(|T_u|)$.

Contracting a path P can be done in $O(|P|^3)$, by taking all possible pairs of vertices and edges verifying if a contraction operation can be done between them. Hence, step 4 can be executed in $O(|T_u|^3)$.

Lastly, step 5 can be executed in $O(1)$.

Since the size and the number of elements in $\mathrm{FSC}_{\mathrm{k,q}}(u)$ is bounded by $f'(k,q,k') = O(kqk')$ and $f(k,q,k') = O((60kqk')^{45kqk'})$ respectively from Lemma 28, we get the result. That is, the complexity of procedure *ForgetNode* is bounded by, if $q < \infty$:

$$f_f(k,q,k') = O\left(f(k,q,k') \cdot f'(k,q,k')^3\right) = O\left((kqk')^3(60kqk')^{45kqk'}\right).$$

If $q = \infty$ then the complexity of procedure *ForgetNode* is bounded by:

$$f_f(k,k') = O\left(f(k,k') \cdot f'(k,k')^3\right) = O\left((kk')^3(15kk')^{45kqk'}\right).$$

□

Procedure *JoinNode*

Let v be a join node of D, let u, w be its children, let $\mathrm{FSC}_{\mathrm{k,q}}(u)$ be a full set of characteristics of A_u restricted to X_u, and $\mathrm{FSC}_{\mathrm{k,q}}(w)$ a full set of characteristics of A_w restricted to X_w.

Remark 4. Procedure *JoinNode* tries to merge the (k,q)-characteristics for X_u and X_w that share a same structure, in contrast with the procedure Join Node from Section 4.3 that merges labeled partitioning trees for A_u and A_w that are isomorphic.

The *skeleton* $\mathrm{Sk}(C)$ of $C = ((T', r', \sigma'), \ell', K', \mathrm{dist}', out', branch', father')$ is the tree obtained from T' by contracting all vertices that are not in K' (these vertices have degree two, thus the notion of contraction is well defined). Therefore, $V(\mathrm{Sk}(C)) = K'$. Two partitioning trees (T, r, σ) and (T', r', σ') are *isomorphic* if there is an one-to-one function $\varphi : V(T) \to V(T')$ preserving the edges, such that $\varphi(r) = r'$, and moreover, $\sigma'(\varphi(f)) = \sigma(f)$ for any leaf f of T.

The *structure* $\mathrm{Struct}(C)$ of a characteristic C is the partitioning tree obtained from $\mathrm{Sk}(C)$ by contracting all its vertices with degree two, different from the root. That is, we only keep branching nodes of T in $\mathrm{Struct}(C)$, while keeping the same root and the same mapping over the leaves of the tree. For any characteristic $C_u = ((T_u, r_u, \sigma_u), \ell_u, K_u, \mathrm{dist}_u,$

$out_u, branch_u, father_u) \in \mathrm{FSC}_{\mathrm{k,q}}(u)$ and $C_w = ((T_w, r_w, \sigma_w), \ell_w, K_w, \mathrm{dist}_w, out_w, branch_w, father_w) \in \mathrm{FSC}_{\mathrm{k,q}}(w)$, with isomorphic structures and with $\mathrm{dist}_w = 0$ (if both have distance non-zero we do not do the procedure *JoinNode* with C_u and C_w, since the roots r_u and r_v come from different branches of the partitioning tree), Procedure *JoinNode* proceeds as follows, repeating the five steps below, for any possible execution of Step 2. Roughly, it merges C_u and C_w to obtain $C_v = ((T_v, r_v, \sigma_v), \ell_v, K_v, \mathrm{dist}_v, out_v, branch_v, father_v)$.

1. *identifying the structures:*

   To obtain T_v, we start by a copy of T_u and a copy of T_w.

   Then, for any vertex $t' \in V(\mathrm{Struct}(C_u)) = V(\mathrm{Struct}(C_w))$, let t_u be the corresponding vertex in T_u, and t_w be the corresponding vertex in T_w.

   In T_v, we identify t_u with t_w.

   Note that r_u and r_w are identified, let r_v be the resulting vertex.

   Then, since $\mathrm{dist}_w = 0$, we set $\mathrm{dist}_v \leftarrow \mathrm{dist}_u$.

2. *merging the paths:*

   For any $\{x, y\} \in E(\mathrm{Struct}(C_u))$, let x and y be vertices of T_u resulting of the identification of x_u with x_w and y_u with y_w respectively.

   Currently in T_v, there are two paths between x and y, a path P_u (initially a path of T_u) and a path P_w (initially a path of T_w), these paths are vertex-disjoint except for x and y. Since internal vertices of P_u and P_w do not belong to $V(\mathrm{Struct}(C_u)$ nor to $V(\mathrm{Struct}(C_w)$, any internal vertex (resp., edge) of both these paths defines the same partition $\mathcal{P}$ of X_v.

   Then, we replace P_u and P_w in T_v with a merging of P_u and P_w using the function $F : (i, j) \to H_\Phi(i, j, \mathcal{P})$.

3. *update of K_v:*

   Roughly, K_v is obtained by taking $K_u \cup K_w$ and some other vertices.

   Formally, starting from $K_v = \emptyset$.

   For any vertex x_v in T_v that results from the identification of $x_u \in V(T_u)$ and $x_w \in V(T_w)$ we set $K_v \leftarrow K_v \cup \{x_v\}$. In other words, x_u and x_w are either leaves of T_u and T_w or they are branching nodes of T_u and T_w, consequently they x_v is either a leaf or a branching node of T_v.

   For any other vertex x_v in $V(T_v)$, assume that x_v is obtained through the merging o a path P_u of T_u and a path P_w of P_w, as described in the step "merging of paths". Let x_u be the vertex of the extension of P_u used to generate x_v and x_w be the vertex of the extension of P_w used to generate x_v during the merging of P_u and P_w. Then, if $x_u \in K_u$ or $x_w \in K_w$, then we set $K_v \leftarrow K_v \cup \{x_v\}$.

4. *update of labels:*

   For any $x_v \in V(\mathrm{cp}(T_v))$:

$$\begin{aligned} branch_v(x_v) &\leftarrow \max\{branch_u(x_u), branch_w(x_w)\}, \\ out_v(x_v) &\leftarrow \max\{out_u(x_u), out_w(x_w)\}, \\ father_v(x_v) &\leftarrow \max\{father_u(x_u), father_w(x_w)\}. \end{aligned}$$

For every $x_v \in V(\mathrm{cp}(T_v))$, if $branch_u(x_u) = branch_w(x_w) = 0$ and $father_u(x_u) = father_w(x_w) = 1$, then $branch_v(x_v) \leftarrow 1$.

For every $x_v \in V(T_v)$ such that x_v is a leaf, $\sigma_v(x_v) \leftarrow \sigma_u(x_u)$.

5. *contracting the paths:*

   $\forall x, y \in K_v$ and path P between x and y such that no internal vertices of P are in K_v, $P \leftarrow \mathrm{Contr}(P)$.

6. *update of* $\mathrm{FSC}_{\mathrm{k,q}}(v)$*:*

   $\mathrm{brheight}_T(r_v)$ is computable thanks to out_v and $branch_v$. If $\mathrm{dist}_v + \mathrm{brheight}_T(r_v) \leq q$ and $\ell_v(t) \leq k$ for any internal vertex $t \in V(T_v)$, and $\ell_v(e) \leq k$ for any edge $e \in E(T_v)$, then $\mathrm{FSC}_{\mathrm{k,q}}(v) \leftarrow \mathrm{FSC}_{\mathrm{k,q}}(v) \cup \{C_v\}$.

The rest of this section is dedicated to proving Lemma 37.

Lemma 37. JoinNode *computes a full set of* (k, q)*-characteristics of* A_v *restricted to* X_v*.*

Proof. Since $A_v = A_u \cup A_w$, A_v admits a q-branched partitioning tree with Φ-width at most k only if A_u and A_w do. Therefore, we can assume that $\mathrm{FSC}_{\mathrm{k,q}}(u) \neq \emptyset$ and $\mathrm{FSC}_{\mathrm{k,q}}(w) \neq \emptyset$, otherwise, A_v does not admit a q-branched partitioning tree with Φ-width at most k, and $\mathrm{FSC}_{\mathrm{k,q}}(v) = \emptyset$.

To prove that the set $\mathrm{FSC}_{\mathrm{k,q}}(v)$ is a full set of characteristics, we take any q-branched partitioning tree $(T_v^\circ, r_v^\circ, \sigma_v^\circ)$ of A_v with Φ width at most k and show that after procedure *JoinNode* finishes we have that $\mathrm{Char}((T_v^\circ, r_v^\circ, \sigma_v^\circ), X_v) \in \mathrm{FSC}_{\mathrm{k,q}}(v)$.

Roughly, we show that there is there is a particular execution of step 2 of procedure *JoinNode* with two characteristics, $C_u \in \mathrm{FSC}_{\mathrm{k,q}}(u)$ and $C_w \in \mathrm{FSC}_{\mathrm{k,q}}(w)$, resulting in C_v such that $C_v = \mathrm{Char}((T_v^\circ, r_v^\circ, \sigma_v^\circ), X_v))$. A scheme of the proof of Lemma 37 can be found in Figure 4.8.

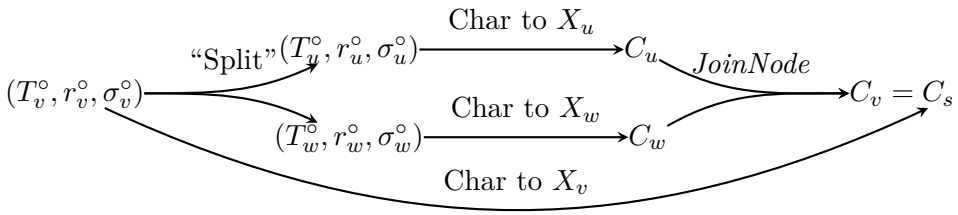


Figure 4.8: Scheme of proof of Lemma 37.

Let $(T_v^\circ, r_v^\circ, \sigma_v^\circ)$ be any q-branched partitioning tree of A_v with Φ width at most k. Let $C_s = \mathrm{Char}((T_v^\circ, r_v^\circ, \sigma_v^\circ), X_v)$. That is, $C_s = ((T_s, r_s, \sigma_s), \ell_s, K_s, \mathrm{dist}_s, out_s, branch_s, father_s)$ is the (k, q)-characteristic of $(T_v^\circ, r_v^\circ, \sigma_v^\circ)$ restricted to X_v.

Let T_v' be the smallest subtree of T_v° spanning all leaves of T_v° that map elements of X_v. Let r_v' be the vertex of T_v' that is closest to r_v° in T_v°. That is, T_v' and r_v' are the tree and the vertex obtained with the first step of procedure $\mathrm{Char}((T_v^\circ, r_v^\circ, \sigma_v^\circ), X_v)$.

Let T_u° (resp. T_w°) be the smallest subtree of T_v° spanning all leaves of T_v° that map elements of A_u (resp. A_w) and let r_u° (resp. r_w°) be the vertex of T_u° (resp. T_w°) that is closest to r_v° in T_v°.

Since $(D, \mathcal{X})$ is a nice decomposition of A we have that $A_u \cap A_w = X_v$. Therefore, we have that $V(T_u^\circ) \cap V(T_w^\circ) = V(T_v')$ and $E(T_u^\circ) \cap E(T_w^\circ) = E(T_v')$. Moreover, since

$A_v = A_u \cup A_w$, we have that either ($r_u^\circ = r_v^\circ$ and $r_w^\circ = r'_v$) or ($r_w^\circ = r_v^\circ$ and $r_u^\circ = r'_v$). W.l.o.g. assume that $r_u^\circ = r_v^\circ$ and $r_w^\circ = r'_v$.

Then, $(T_u^\circ, r_u^\circ, \sigma_u)$, where σ_u is the restriction of σ_v over A_u, is a q-branched partitioning tree with Φ width at most k for A_u and $(T_w^\circ, r_w^\circ, \sigma_w)$, where σ_w is the restriction of σ_v over A_w, is a q-branched partitioning tree with Φ width at most k for A_w.

Let $C_u = \text{Char}((T_u^\circ, r_u^\circ, \sigma_u), X_u)$ and $C_w = \text{Char}((T_w^\circ, r_w^\circ, \sigma_w), X_w)$, such that $C_u = ((T_u, r_u, \sigma_u), \ell_u, K_u, \text{dist}_u, \mathit{out}_u, \mathit{branch}_u, \mathit{father}_u)$ and such that $C_w = ((T_w, r_w, \sigma_w), \ell_w, K_w, \text{dist}_w, \mathit{out}_w, \mathit{branch}_w, \mathit{father}_w)$. Since $(T_u^\circ, r_u^\circ, \sigma_u)$ and $(T_w^\circ, r_w^\circ, \sigma_w)$ are q-branched and with Φ width at most k, we have that $C_u \in \text{FSC}_{\text{k,q}}(u)$ and $C_w \in \text{FSC}_{\text{k,q}}(w)$. We want to show that there is an execution of procedure *JoinNode* on C_u and C_w generating C_v such that $C_v = C_s$.

In order to do that, we must first show that $\text{Struct}(C_u)$ and $\text{Struct}(C_w)$ are isomorphic and that either $\text{dist}_u = 0$ or $\text{dist}_w = 0$.

Claim 10. *Procedure* JoinNode *can be applied to* C_u *and* C_w*. That is,* $\text{Struct}(C_u)$ *and* $\text{Struct}(C_w)$ *are isomorphic and either* $\text{dist}_u = 0$ *or* $\text{dist}_w = 0$.

From the fact that $X_v = X_u = X_w$, we have that, in the first step of $\text{Char}((T_u^\circ, r_u^\circ, \sigma_u), X_u)$ and $\text{Char}((T_w^\circ, r_w^\circ, \sigma_w), X_w)$ the smaller subtree of T_u° and T_w° spanning all vertices in $X_u = X_w = X_v$ is T'_v.

Since $r'_v = r_w^\circ$, from step 2 of the procedure $\text{Char}((T_w^\circ, r_w^\circ, \sigma_w), X_w)$ we have that $\text{dist}_w = 0$.

We need to show that $\text{Struct}(C_u)$ and $\text{Struct}(C_w)$ are isomorphic. From the definition of structure, $\text{Struct}(C_u)$ is obtained from $(\text{Sk}(C_u), r_u, \sigma_u)$ by taking contracting all vertices of degree two that are different from the root from $\text{Sk}(C_u)$. On the other hand, $\text{Sk}(C_u)$ is the tree obtained from T_u by contracting all vertices that are not in K_u. Let $\text{Struct}(C_u) = (T_u^r, r_u, \sigma_u)$ and $\text{Struct}(C_u) = (T_w^r, r_w, \sigma_w)$, where T_u^r and T_w^r are the trees obtained by contracting all vertices of degree two different from the root (r_u and r_w respectively) from $\text{Sk}(C_u)$ and $\text{Sk}(C_w)$ respectively.

Since, r'_v is the vertex of T'_v that is closest to $r_u^\circ = r_v^\circ$ in T_u°, we have that $r_u = r'_v$. Then, since $r'_v = r_w$ we have that $r_u = r_w$.

Note that leaves of T'_v cannot be contracted, since they have degree one, in order to obtain T_u^r and T_w^r. In other words, if x is a leaf of T'_v we have that $x \in T_u^r$ and $x \in T_w^r$. Therefore, from the step one of Char procedure we have that, for any leaf $x \in V(T'_v)$, $\sigma_u(x) = \sigma_v^\circ(x) = \sigma_w(x)$.

Now we only need to show that the trees T_u^r and T_w^r are isomorphic. In fact, we shall show that $T_u^r = T_w^r$.

Note that, from the steps one and six of the Char procedure, T_u (resp. T_w) is obtained from T'_v by applying some contraction operations on vertices that are not in K_u (resp. K_w), since they have degree two this is well defined. On the other hand, $\text{Sk}(C_u)$ (resp. $\text{Sk}(C_w)$) is the tree obtained from T_u by contracting all vertices with degree two that are not in K_u (resp. K_w). Hence, $\text{Sk}(C_u)$ (resp. $\text{Sk}(C_w)$) can be obtained from T'_v by contracting all vertices that are not in K_u (resp. K_w). Then, T_u^r (resp. T_v^r) is obtained from $\text{Sk}(C_u)$ (resp. $\text{Sk}(C_w)$) by contracting all vertices of degree two which are different from the root $r_u = r'_v$ (resp. $r_w = r'_v$).

Since, both T_u^r and T_w^r are obtained from T'_v by contracting some vertices with degree two, we only need to show that the same vertices of T'_v are contracted to obtain T_u^r and T_v^r. That is, we only need to show that $V(T_u^r) = V(T_v^r)$. In order to do that consider the

following cases for $x \in V(T'_v) \setminus \{r'_v\}$: (1) $x \notin K_u \cup K_w$, (2) $x \in K_u \setminus K_w$, (3) $x \in K_w \setminus K_u$ and (4) $x \in K_u \cap K_w$.

Case 1: If $x \notin K_u \cup K_w$ then x has degree two in T'_v and it is contracted to obtain T^r_u and T^r_w. Hence, $x \notin V(T^r_u)$ and $x \notin V(T^r_w)$.

Case 2: If $x \in K_u \setminus K_w$, then x is not a leaf of T'_v, is not the parent of a leaf in T'_v, is not a branching node of T'_v and it is not a branching node of $\mathrm{cp}(T^\circ_w)$, otherwise x would be in K_w. Therefore, x has degree two in T'_v. Since, x is not r'_v we have that x is contracted in the process to obtain T^r_u from $\mathrm{Sk}(C_u)$. In other words, $x \notin V(T^r_u)$. On the other hand, $x \notin K_w$, hence $x \notin V(\mathrm{Sk}(C_w))$ and, consequently, $x \notin V(T^r_w)$.

Case 3: This case is similar to Case 2 with the role of K_u and K_w reversed. If $x \in K_w \setminus K_u$, then x is not a leaf of T'_v, is not the parent of a leaf in T'_v, is not a branching node of T'_v and it is not a branching node of $\mathrm{cp}(T^\circ_u)$, otherwise x would be in K_u. Therefore, x has degree two in T'_v. Since, x is not r'_v we have that x is contracted in the process to obtain T^r_w from $\mathrm{Sk}(C_w)$. In other words, $x \notin V(T^r_w)$. On the other hand, $x \notin V(\mathrm{Sk}(C_u))$ and, consequently $x \notin V(T^r_u)$.

Case 4: If $x \in K_u \cap K_w$, then $x \in \mathrm{Sk}(C_u)$ and $x \in \mathrm{Sk}(C_w)$. Moreover, either x is a leaf of T'_v, is the parent of a leaf in T'_v, is a branching node of T'_v, or is a branching node of $\mathrm{cp}(T^\circ_u)$ and $\mathrm{cp}(T^\circ_w)$.

If x is a leaf of T'_v, then x has degree one in T'_v and, hence, $x \in V(T^r_u)$ and $x \in V(T^r_w)$.

If x is not a leaf of T'_v, then x has degree two in $\mathrm{Sk}(C_u)$ if an only if x has degree two in $\mathrm{Sk}(C_w)$. This is due to the fact that, by contracting vertices of degree two of a tree, we do not change the degrees of the remainder vertices.

This shows that $V(T^r_u) = V(T^r_w)$. From the fact that T^r_u and T^r_w are both obtained from T'_v by contraction of vertices of degree two we get the result. That is, $\mathrm{Struct}(C_u)$ and $\mathrm{Struct}(C_w)$ are isomorphic.

Therefore, the there is an execution of procedure *JoinNode* were C_u is merged with C_w. Let $C_v = ((T_v, r_v, \sigma_v), \ell_v, K_v, \mathrm{dist}_v, \mathit{out}_v, \mathit{branch}_v, \mathit{father}_v)$ be the result of a particular execution of procedure *JoinNode* on C_u and C_w, that will be explained latter in this proof.

We want to show that $C_v = C_s$.

How C_v is obtained.

We need to specify how procedure *JoinNode* merges the paths in T_u with the paths in T_w. That is, we need to specify how step "merging the paths" proceeds to merge the paths of T_u and T_w to obtain T_v. In order to do that, we first show how paths in T°_v with labels given by the function Φ can be seen as mergings of the corresponding paths in T°_u and T°_w.

Let $\{x, y\}$ be any edge in $E(\mathrm{Struct}(C_u)) = E(\mathrm{Struct}(C_w))$. We have that, in T'_v, all the internal vertices and edges on the path $P'_v(x, y)$ between x and y define the same partition $\mathcal{T}$ of X_v, since the vertices have degree two. Let $P^\circ_v(x, y)$, $P^\circ_u(x, y)$ and $P^\circ_w(x, y)$ be the paths between x and y in T°_v, T°_u and T°_w respectively. Note that the internal vertices and edges of these paths define the same partition $\mathcal{T}$ of X_v.

For any internal vertex or edge x' of $P^\circ_v(x, y)$ let $\mathcal{P}(x')$ be the partition of A_v defined by x'. Then, in T°_u and in T°_w, the vertex or edge x' defines the partition $\mathcal{P}(x') \cap A_u$ of A_u and $\mathcal{P}(x') \cap A_w$ of A_w respectively. Since Φ is compatible with $(D, \mathcal{X})$, we have that there is a function H_Φ such that for any partition $\mathcal{P}$ of A_v it is true that $\Phi_{A_v}(\mathcal{P}) =$

$H_\Phi(\Phi_{A_u}(\mathcal{P} \cap A_u), \Phi_{A_w}(\mathcal{P} \cap A_w), \mathcal{P} \cap X_v)$. Therefore, for any internal vertex or edge x' of $P_v^\circ(x,y)$ we have that $\Phi_{A_v}(\mathcal{P}(x')) = H_\Phi(\Phi_{A_u}(\mathcal{P}(x') \cap A_u), \Phi_{A_w}(\mathcal{P}(x') \cap A_w), \mathcal{T})$. Hence, the labeled path $P_v^\circ(x,y)$ with labels given by the function Φ can be obtained by the merging of the path $P_u^\circ(x,y)$ and $P_w^\circ(x,y)$, both with labels given by Φ, under the function $F(i,j) = H_\Phi(i,j,\mathcal{T})$. Since $|V(P_u^\circ(x,y))| = |V(P_w^\circ(x,y))|$, the extensions of $P_u^\circ(x,y)$ and $P_w^\circ(x,y)$ used in the merging are simply $P_u^\circ(x,y)$ and $P_w^\circ(x,y)$ themselves. In other words, we can merge the paths $P_u^\circ(x,y)$ and $P_w^\circ(x,y)$ under the function F to obtain the path $P_v^\circ(x,y)$.

Let $P_u(x,y)$ (resp. $P_w(x,y)$) be the path between x and y in T_u (resp. T_v). In step "merging the paths" of procedure *JoinNode*, the path $P_u(x,y)$ is merged with the path $P_w(x,y)$. From the Char procedure, we have that $P_u(x,y)$ (resp. $P_w(x,y)$) is obtained from $P_u^\circ(x,y)$ (resp. $P_w^\circ(x,y)$) by applying some contraction operations. Then, let $P_v(x,y)$ be a merging of $P_u(x,y)$ with $P_w(x,y)$ under the function F such that $P_v^\circ(x,y)$ *respects*[2] $P_v(x,y)$. Roughly, this means that the vertices and edges of $P_v(x,y)$, which is a merging of $P_u(x,y)$ and $P_w(x,y)$ under the function F, have "equivalent" vertices in $P_v^\circ(x,y)$, which is a merging of $P_u^\circ(x,y)$ and $P_w^\circ(x,y)$ under the same function F.

Since procedure *JoinNode* tries all possible ways of merging $P_u(x,y)$ and $P_w(x,y)$ for all $\{x,y\} \in E(\text{Struct}(C_u))$, we have that there is an execution of procedure *JoinNode* where for all $\{x,y\} \in E(\text{Struct}(C_u))$ the path $P_v(x,y)$ obtained through the merging of $P_u(x,y)$ and $P_w(x,y)$ under F is respected by $P_v^\circ(x,y)$.

Let T_v be the tree obtained by such execution of procedure *JoinNode*.

To show that C_v, obtained through this particular execution of procedure *JoinNode*, is equal to $C_s = \text{Char}((T_v^\circ, r_v^\circ, \sigma_v^\circ), X_v)$, we start by showing that T_v is isomorphic to T_s.

Claim 11. *T_v and T_s are isomorphic and the labels of correspondent vertices of T_v and T_s are the same. That is, $\ell_v(t_v) = \ell_s(t_s)$ for all internal vertex or edge t_v of T_v that has a corresponding vertex or edge t_s in T_s.*

In the following consider that the labels of a tree are given by either the associated function ℓ or by Φ if the tree has no associated function ℓ. That is, the labels of T_u, T_w, T_s and T_v are given by ℓ_u, ℓ_w, ℓ_s and ℓ_v, while the labels of T_v°, T_v', T_u° and T_w° are given by the function Φ.

Note that from the fact that T_v is obtained by merging paths $P_u(x,y)$ and $P_w(x,y)$ for every edge $\{x,y\} \in E(\text{Struct}(C_u))$ we have that the tree obtained by contracting all vertices of degree two in T_v is isomorphic to the tree in $\text{Struct}(C_u)$.

As explained above, for any edge $\{x,y\} \in E(\text{Struct}(C_u))$ to obtain T_v we first merge the paths $P_u(x,y)$ and $P_w(x,y)$ obtaining $P_v(x,y)$ and then we apply some contraction operations on $P_v(x,y)$ (step "contracting the paths") obtaining $P_v^c(x,y)$. More precisely, let $(x_1, \ldots, x_z)$ be the sequence of vertices in the path from x to y in the tree obtained immediately after step "merging the paths" of procedure *JoinNode* that are in K_v. That is, $P_v(x,y)$ can be written as $P_v(x,x_1) \odot P_v(x,x_2) \odot \cdots \odot P_v(x_z,y)$ where $x_i \in K_v$, for $1 \leq i \leq z$. Then, $P_v(x,y)$ is replaced by $\text{Contr}(P_v(x,x_1)) \odot \text{Contr}(P_v(x_1,x_2)) \odot \cdots \odot \text{Contr}(P_v(x_z,y))$. Let $P_v^c(x,y) = \text{Contr}(P_v(x,x_1)) \odot \text{Contr}(P_v(x,x_2)) \odot \cdots \odot \text{Contr}(P_v(x_z, y))$. Hence, for each $\{x,y\} \in E(\text{Struct}(C_u))$ we have that $P_v^c(x,y)$ is the path in T_v between x and y.

Note that, $K_s = K_u \cup K_w$. Since, the only case where $x \in K_s$ and $x \notin K_u$ is when x is not a leaf of T_v', nor the parent of a leaf of T_v', nor a branching node of $\text{cp}(T_u^\circ)$, but it

[2]For the definition of "respect" see "Merging of labeled Paths" in Section 4.4.

is a branching node of $\mathrm{cp}(T_v^\circ)$. Therefore, x is is a branching node of $\mathrm{cp}(T_w^\circ)$ and, hence, $x \in K_w$.

On the other hand, the path $P_s(x,y)$ between x and y in C_s is obtained from $P_v^\circ(x,y)$ by applying some contraction operations. That is, let $(x_1^s, \ldots, x_z^s)$ be the sequence of vertices in the path from x to y in T_v° that are in K_s, then $P_s(x,y)$ can be written as $\mathrm{Contr}(P_v^\circ(x,x_1^s)) \odot \mathrm{Contr}(P_v^\circ(x_1^s,x_2^s)) \odot \cdots \odot \mathrm{Contr}(P_v^\circ(x_z^s,y))$.

Then, this proof essentially follows from Lemma 25 when applied to $P_v^\circ(x,y)$ and $P_v(x,y)$. That is, we consider $P_v^\circ(x,y)$ as M where P and Q are $P_u^\circ(x,y)$ and $P_w^\circ(x,y)$, respectively. $P_v(x,y)$ takes the role as M^c where P^c and Q^c are $P_u(x,y)$ and $P_w(x,y)$ respectively. Finally, we set K_p as the set K_u and K_q as the set K_w. Then, Lemma 25 guarantees that $\mathrm{Contr}(P_v(x,x_1)) = \mathrm{Contr}(P_v^\circ(x,x_1^s))$, $\mathrm{Contr}(P_v(x_z,y)) = \mathrm{Contr}(P_v^\circ(x_z^s,y))$ and, for all $1 \leq i \leq z-1$, $\mathrm{Contr}(P_v(x_i,x_{i+1})) = \mathrm{Contr}(P_v^\circ(x_i^s,x_{i+1}^s))$. Therefore, since $P_s(x,y) = \mathrm{Contr}(P_v^\circ(x,x_1^s)) \odot \mathrm{Contr}(P_v^\circ(x_1^s,x_2^s)) \odot \cdots \odot \mathrm{Contr}(P_v^\circ(x_z^s,y))$ and $P_v^c(x,y) = \mathrm{Contr}(P_v(x,x_1)) \odot \mathrm{Contr}(P_v(x,x_2)) \odot \cdots \odot \mathrm{Contr}(P_v(x_z,y))$, we get the result. That is, $P_v^c(x,y) = P_s(x,y)$.

Therefore T_v and T_s are isomorphic and for all internal vertices or edges t_v of T_v with a correspondent vertex or edge t_s of T_s we have that $\ell_v(t_v) = \ell_s(t_s)$.

Since T_v and T_s are isomorphic, for every vertex $x_s \in T_s$, let x_v be its correspondent in T_v. To make the rest of this proof easier to read, we abuse the notation to say that $x_s = x_v$. We now prove that $r_v = r_s$ and that $K_v = K_s$.

Claim 12. *$r_v = r_s$ and $K_v = K_s$.*

From step "identifying the structures" we have that $r_v = r_u$. We have that $r_u = r'_v$ from step one of the Char procedure to obtain C_u. Then, by the fact that $r_s = r'_v$ from step one of the Char procedure to obtain C_s, we have that $r_s = r_v$.

Let x_s be a vertex of T_s that is not the root of T_s. Since T_v is isomorphic to T_s, set x_v be the corresponding vertex of x_s in T_v. To show that $K_s = K_v$, there are a few cases to consider: (1) x_s is a leaf of T'_v, (2) x_s is a branching node of T'_v, (3) x_s is the parent of a leaf in T'_v, (4) x_s is a branching node of $\mathrm{cp}(T_v^\circ)$, (5) otherwise. That is, cases (1) to (4) are all the cases when $x_s \in K_s$, while, in case (5), $x_s \notin K_s$. We want to show that $x_s \in K_s$ if and only if $x_v \in K_v$.

Case (1): If x_s is a leaf of T'_v, then x_s has degree one in T'_v. Therefore, x_s is a vertex of $\mathrm{Struct}(C_u)$ and $\mathrm{Struct}(C_w)$. Then, during step "identifying the structures" x_v is obtained through the identification of $x_s \in V(\mathrm{Struct}(C_u))$ and $x_s \in V(\mathrm{Struct}(C_w))$. Hence, during step "update of K_v" we have that x_v is put into K_v. That is, $x_v \in K_v$.

Case (2): If x_s is a branching node of T'_v, then x_s has degree at least three in T'_v. Therefore, x_s is a vertex of $\mathrm{Struct}(C_u)$ and $\mathrm{Struct}(C_w)$. Then, during step "identifying the structures" x_v is obtained through the identification of $x_s \in V(\mathrm{Struct}(C_u))$ and $x_s \in V(\mathrm{Struct}(C_w))$. Hence, during step "update of K_v" we have that x_v is put into K_v. That is, $x_v \in K_v$.

Case (3): If x_s is the parent of a leaf in T'_v and its not a branching node of T'_v, then x_s has degree two in T'_v. Since x_s has degree two in T'_v we have that $x_s \notin V(\mathrm{Struct}(C_u))$. Therefore, x_v obtained during step "merging the path". That is, x_v is obtained through the merging of two vertices $x_u \in T_u$ and $x_w \in T_w$.

Let $\{x,y\} \in E(\mathrm{Struct}(C_u))$ be two vertices such that x_s is an internal vertex on the path $P_v^\circ(x,y)$ from x to y in T_v°. Then, x_v is obtained through the merging

$P_u(x,y)$ and $P_w(x,y)$. Let $P_v(x,y)$ be the resulting path. Recall that $P_v^\circ(x,y)$ can be written as a merging of the paths $P_u^\circ(x,y)$ and $P_w^\circ(x,y)$. From the construction of C_v we have that $P_v(x,y)$ is respected by $P_v^\circ(x,y)$.

Then, $x_s \in V(T_v^\circ)$ is the result of matching the vertex $x_u = x_s \in T_u^\circ$ with the vertex $x_w = x_s \in T_w^\circ$. On the other hand, since $P_v(x,y)$ is respected by $P_v^\circ(x,y)$, we have that x_v is obtained through the merging of $x_u \in T_u$ and $x_w \in T_w$.

From the Char procedure, since x_s is the parent of a leaf in T'_v, we have that $x_u \in K_u$ and $x_w \in K_w$. Then, since $x_u \in K_u$ and $x_w \in K_w$, step "update of K_v" ensures that $x_v \in K_v$.

Case (4): If x_s is a branching node of $\mathrm{cp}(T_v^\circ)$ and x_s is not the parent of a leaf in T'_v and its not a branching node of T'_v, then x_s has degree two in T'_v and x_s has a non leaf child in T'_v. Since x_s is a branching node of $\mathrm{cp}(T_v^\circ)$ and has degree two in T'_v, x_s has at least one non leaf child in $V(T_v^\circ) \setminus V(T'_v)$. Let x'_s be this child. Therefore, from the fact that $V(T_v^\circ) = V(T_u^\circ) \cup V(T_w^\circ)$ we have that x'_s is either in $V(T_u^\circ)$ or in $V(T_w^\circ)$. W.l.o.g. assume that $x'_s \in V(T_u^\circ)$, hence x'_s is a branching node of $\mathrm{cp}(T_u^\circ)$. Consequently, from the Char procedure to obtain C_v, $x'_s \in K_u$. Note that, from step "update of K_v" if x_v is obtained by merging (or matching) a vertex x_u and x_w, then $x_v \in K_v$ if either $x_u \in K_u$ or $x_w \in K_w$. Then, the rest of this proof is similar to Case (3) and thus omitted.

Case (5): If $x_s \notin K_s$, then x_s has degree two in T'_v, $x_s \notin K_u$ and $x_s \notin K_w$. Hence, following the same reasoning in Case (3), we have that x_v is obtained through the merging (or matching) of $x_u \notin K_u$ and $x_w \notin K_w$. Therefore, from step "update of K_v" we have that $x_v \notin K_v$.

Therefore, $x_v \in K_v$ if and only if $x_s \in K_s$.

Considering the previous claims, we only need to prove the following claim in order to show that $C_v = C_s$.

Claim 13. $\mathrm{dist}_s = \mathrm{dist}_v$.
For all leaves $x_s \in V(T_s)$ we have that $\sigma(x_v) = \sigma(x_s)$.
For all vertices $x_s \in V(\mathrm{cp}(T_s))$ we have that:

$$\begin{aligned} out_s(x_s) &= out_v(x_v); \\ branch_s(x_s) &= branch_v(x_v); \\ father_s(x_s) &= father_v(x_v). \end{aligned}$$

From step "identifying the structures" we have that $\mathrm{dist}_v = \mathrm{dist}_u$. Note that dist_s is the number of branching nodes of T_v° between r_v° and r'_v. On the other hand, dist_u is the number of branching nodes of T_u° between r_u° and r'_v. Since $r_u^\circ = r_v^\circ$ and the path $P_v^\circ(r_v^\circ, r'_v)$, the path between r_v° and r'_v in T_v°, is equal to the path $P_u^\circ(r_u^\circ, r'_v)$, the path between r_u° and r'_v in T_u°, we get that $\mathrm{dist}_u = \mathrm{dist}_s$. Therefore, $\mathrm{dist}_v = \mathrm{dist}_s$.

Let x_s be a leaf of T_s and x_v be its corresponding vertex in T_v. Since x_s is a leaf of T_s, it is also a leaf in T'_v. Note that, from the definition of $(T_v^\circ, r_v^\circ, \sigma_v^\circ)$ and $(T_u^\circ, r_u^\circ, \sigma_u^\circ)$, we have that $\sigma_v^\circ(x_s) = \sigma_u^\circ(x_s)$. On the one hand, we have that, from the fact that $C_s = \mathrm{Char}((T_v^\circ, r_v^\circ, \sigma_v^\circ), X_v)$, $\sigma_s(x_s) = \sigma_v^\circ(x_s)$. On the other hand, from the $C_u = \mathrm{Char}((T_u^\circ, r_u^\circ, \sigma_u^\circ), X_u = X_v)$, we have that $\sigma_u(x_s) = \sigma_u^\circ(x_s)$. Therefore, $\sigma_u(x_s) = \sigma_s(x_s)$. Note that, since x_s is a leaf in T'_v, $x_s \in V(\mathrm{Struct}(C_u) = V(\mathrm{Struct}(C_w)))$.

Therefore, x_v is obtained through the identification of $x_s = x_u \in V(\text{Struct}(C_u)$ and $x_s = x_w \in V(\text{Struct}(C_w))$. Then, during step "update of labels" of procedure *JoinNode*, $\sigma_v(x_v)$ is set to be $\sigma_u(x_u = x_s)$. Hence, we get the result. That is, for every leaf $x_s \in V(T_s)$ we have that $\sigma(x_v) = \sigma(x_s)$.

Let x_s be a vertex of $\text{cp}(T_s)$. That is, x_s is an internal vertex of T_s. Let x_v be its correspondent vertex in T_v with x_u and x_w being the vertices of T_u and T_w used to create x_v. In other words, x_v is obtained from the merging of x_u and x_w or from the identification of x_u with x_w.

In C_s, $out_s(x_s)$ is the maximum number of branching nodes in a path of T_v° between x_s and a leaf in $V(T_v^\circ) \setminus V(T'_v)$ with no internal vertices belonging to $V(T'_v)$. Then, let $P(x_s, l)$ be any path of T_v° between x_s and a leaf in $V(T_v^\circ) \setminus V(T'_v)$ such that the number of branching nodes in this path is maximum and with no internal vertex of $P(x_s, l)$ belonging to T'_v. We have that $V(P(x_s, l)) \setminus \{x_s\}$ is entirely contained in $V(T_v^\circ) \setminus V(T'_v)$.

Since T_u° and T_w° are subtrees of T_v° such that $V(T_u^\circ) \cup V(T_w^\circ) = V(T_v^\circ)$, we have that either $l \in V(T_u^\circ)$ or $l \in V(T_v^\circ)$, but not both since $l \notin V(T'_v) = V(T_u^\circ) \cap V(T_w^\circ)$. Therefore, $V(P(x_s, l)) \setminus \{x_s\} \subseteq V(T_u^\circ) \setminus V(T'_v)$ or $V(P(x_s, l)) \setminus \{x_s\} \subseteq V(T_w^\circ) \setminus V(T'_v)$. That is, if $l \in V(T_u^\circ)$, then $P(x_s, l)$ is a path of T_u° that starts in x_s does not pass through any vertex in T'_v and ends in l, otherwise $P(x_s, l)$ is a path of T_w° that starts in x_s does not pass through any vertex in T'_v and ends in $l \in V(T_w^\circ)$.

If $l \in V(T_u^\circ)$, from the fact that $C_u = \text{Char}((T_u^\circ, r_u^\circ, \sigma_u^\circ), X_u = X_v)$, then $out_u(x_u) = out_s(x_s)$. If $l \in V(T_w^\circ)$, from the fact that $C_w = \text{Char}((T_w^\circ, r_w^\circ, \sigma_w^\circ), X_w = X_v)$, then $out_w(x_w) = out_s(x_s)$.

Moreover, for all leaves $l \in V(T_u^\circ) \setminus V(T'_v)$, the paths $P(x_s, l)$ in T_u° such that $P(x_s, l)$ has no internal vertex in T'_v are all paths in T_v° that do not pass through any vertex in T'_v. Hence, $out_u(x_u) \leq out_s(x_s)$.

Similarly, for all leaves $l \in V(T_u^\circ) \setminus V(T'_v)$, the paths $P(x_s, l)$ in T_u° such that $P(x_s, l)$ has no internal vertex in T'_v are all paths in T_v° that do not pass through any vertex in T'_v. Hence, $out_w(x_w) \leq out_s(x_s)$.

Therefore, from step "update of labels", we have that $out_v(x_v) = \max\{out_u(x_u), out_w(x_w)\} = out_s(x_s)$.

If $branch_s(x_s) = 1$, then x_s is a branching node of $\text{cp}(T_v^\circ)$. Therefore, either (1) x_s is a branching node in $\text{cp}(T_u^\circ)$, (2) x_s is a branching node in $\text{cp}(T_w^\circ)$, or (3) x_s has exactly one non leaf child in T_u° and exactly one non leaf child in T_w°.

Case 1: If x_s is a branching node in $\text{cp}(T_u^\circ)$, then $x_s \in K_s \cap K_u$. Hence, x_s is not contracted during $\text{Char}((T_u^\circ, r_u^\circ, \sigma_u^\circ), X_u)$ in order to obtain T_u, that is $x_u = x_s \in V(T_u)$.

To obtain T_v we merge the paths in T_u and T_w in such a way that each path merged is respected by the corresponding path in T'_v. Hence, x_v is obtained through the merging of $x_u = x_s$ and x_w. Since, from $\text{Char}((T_u^\circ, r_u^\circ, \sigma_u^\circ), X_u)$, $branch_u(x_u) = 1$ and $branch_v(x_v) = \max\{branch_u(x_u), branch_w(x_w)\}$. Since $branch_w(x_w) \leq 1$, we have that $branch_v(x_v) = branch_s(x_s)$.

Case 2: Similar to case (1), by exchanging the roles of x_u and x_w, and thus omitted.

Case 3: If x_s has exactly one non leaf child in T_u° and exactly one non leaf child in T_w°. Therefore, by the Char procedure, we have that $father_u(x_s = x_u) = 1$ and $father_w(x_s = x_w) = 1$. Then, we have that $father_u(x_u) = 1$ and $father_w(x_w) =$

1. Hence, during step "update of labels", $branch_v(x_v)$ receives the value 1, since $branch_u(x_u) = branch_w(x_w) = 0$ and $father_u(x_u) = father_w(x_w) = 1$.

If $branch_s(x_s) = 0$, then x_s is not a branching node of $\mathrm{cp}(T_v^\circ)$. Hence, x_s is neither a branching node of $\mathrm{cp}(T_u^\circ)$ nor a branching node of $\mathrm{cp}(T_w^\circ)$. Moreover, x_s has no non leaf children in T_u° and no non leaf children in T_w°. Therefore, $branch_u(x_s = x_u) = branch_w(x_s = x_w) = 0$ and $father_u(x_s = x_u) = father_w(x_s = x_w) = 0$. Then, during step "update of labels", $branch_v(x_v) = \max\{branch_u(x_u), branch_w(x_w)\} = 0$.

Hence, we get the result. That is, for all $x_s \in V(\mathrm{cp}(T_s))$, we have that $branch_v(x_v) = branch_s(x_s)$.

Now it remains to show that for all $x_s \in V(\mathrm{cp}(T_s))$ we have that $father_v(x_v) = father_s(x_s)$.

This proof is similar to the proof that if $branch_s(x_s) = branch_v(x_v) = 0$, since $father_s(x_s) = 0$ implies that x_s has no non leaf children in $V(T_v^\circ) \setminus V(T_v')$. Hence, x_s has no non leaf children in $V(T_u^\circ) \setminus V(T_v')$ nor in $V(T_w^\circ) \setminus V(T_v')$.

If $father_s(x_s) = 0$, then x_s is not a branching node of $\mathrm{cp}(T_v^\circ)$. Hence, x_s is neither a branching node of $\mathrm{cp}(T_u^\circ)$ nor a branching node of $\mathrm{cp}(T_w^\circ)$. Moreover, x_s has no non leaf children in T_u° and no non leaf children in T_w°. Therefore, $father_u(x_u) = father_w(x_w) = 0$. Then, during step "update of labels", $father_v(x_v) = \max\{father_u(x_u), father_w(x_w)\} = 0$.

If $father_s(x_s) = 1$, then x_s has either a non leaf child in $V(T_u^\circ) \setminus V(T_v')$ or a non leaf child in $V(T_w^\circ) \setminus V(T_v')$. Hence, $father_u(x_u) = 1$ or $father_w(x_w) = 1$. Then, during step "update of labels", $father_v(x_v) = \max\{father_u(x_u), father_w(x_w)\} = 1$.

Hence, we get the result. That is, for all $x_s \in V(\mathrm{cp}(T_s))$, we have that $father_v(x_v) = father_s(x_s)$.

This concludes the proof that C_v obtained through this execution of procedure *JoinNode* on $C_u \in \mathrm{FSC_{k,q}}(u)$ and $C_w \in \mathrm{FSC_{k,q}}(w)$ is such that $C_v = C_s = \mathrm{Char}((T_v^\circ, r_v^\circ, \sigma_v^\circ), X_v)$.

It remains to show that C_v is put in $\mathrm{FSC_{k,q}}(v)$ by procedure *JoinNode*. By Lemma 27, C_v is a (k,q)-characteristic of A_v restricted to X_v. Hence, $\mathrm{brheight}(r_v) + \mathrm{dist}_v \leq q$ and for all internal vertex or edge x_v of T_v we have that $\ell_v(x) \leq k$. Consequently, during step "update of $\mathrm{FSC_{k,q}}(v)$" we have that $C_v \in \mathrm{FSC_{k,q}}(v)$. Therefore, $\mathrm{FSC_{k,q}}(v)$ is a full set of (k,q)-characteristics of A_v restricted to X_v. □

Theorem 38. *Procedure* JoinNode *computes a full set of (k,q)-characteristics of A_v restricted to X_v in time that does not depend on $|A|$. That is the complexity of procedure* JoinNode *is bounded by a function*

$$f_j(k,q,k') = O\left((60kqk')^{45kqk'} \cdot 2^{O\left(\sqrt{kqk' \cdot \log(kqk')}\right)} \cdot (kqk')^{kqk'}\right), \text{ if } q < \infty,$$

or by a function

$$f_j'(k,k') = O\left((15kk')^{45kk'} \cdot 2^{O\left(\sqrt{kk' \cdot \log(kk')}\right)} \cdot (kk')^{kk'}\right), \text{ if } q = \infty.$$

Proof. From Theorem 37, procedure *JoinNode* computes a full set of (k,q)-characteristics of A_v restricted to X_v. It remains to prove that this can be done time that does not depend on $|A|$.

Assume that $q < \infty$, the case where $q = \infty$ is similar and thus omitted. From the fact that Φ is compatible with $(D, \mathcal{X})$, H_Φ can be computed in constant time.

For each pair of elements $C_u = ((T_u, r_u, \sigma_u), \ell_u, K_u, \mathrm{dist}_u, out_u, branch_u, father_u) \in \mathrm{FSC_{k,q}}(u)$ and $C_w = ((T_w, r_w, \sigma_w), \ell_w, K_w, \mathrm{dist}_w, out_w, branch_w, father_w) \in \mathrm{FSC_{k,q}}(w)$, discovering if their structures are isomorphic can be done in $2^{O(\sqrt{|T_u| \log |T_u|})}$ [BL83].

Then, for each pair of characteristics such that their structures are isomorphic, r_u is the vertex correspondent to r_w and with $\mathrm{dist}_u = 0$ we apply steps 1 to 6 of the *JoinNode* procedure.

Step 1 has complexity bounded by $O(|T_u|)$. Since the merging of two paths P and Q has size limited to $O(|P||Q|)$ by definition of merging, one execution of Step 2 takes at most $O(|T_u||T_w|)$ time. Steps 3 and 4, for each execution of Step 2, takes $O(|T_u||T_w|)$ time.

Since contracting a path P can be done in $O(|P|^3)$, by taking all possible pairs of vertices and edges verifying if a contraction operation can be done between them, Step 5 can be executed in $O((|T_u||T_w|)^3)$, for each execution of Step 2. Lastly, Step 6 can be executed in $O(|T_u||T_w|)$, for each execution of Step 2.

For any pair of paths P and Q merged during step 2 (merging the paths), let $p = \max |P|, |Q|$. Hence, there are at most $O(p^p)$ different ways of merging P and Q. That is, for each edge of P there are at most $|Q|$ edges in Q that is a possible match for P. Therefore, we can upper bound the amount of different possible executions of Step 2 by $O(\max\{|T_u|^{|T_u|}, |T_w|^{|T_w|}\})$. Note that, since $\max\{|T_u|, |T_w|\} \leq f'(k, q, k')$, we have that $O(\max\{|T_u|^{|T_u|}, |T_w|^{|T_w|}\}) \leq O(f'(k, q, k')^{f'(k,q,k'))}$.

Since the size and the number of elements in $\mathrm{FSC_{k,q}}(u)$ is bounded by $f'(k, q, k') = O(kqk')$ and $f(k, q, k') = O((60kqk')^{45kqk'})$ respectively from Lemma 28, we get the result.

That is, complexity of procedure *JoinNode* is bounded by

$$f_j(k, q, k') = O\left(f\left(k, q, k'\right) \cdot 2^{O\left(\sqrt{f'(k,q,k') \cdot \log f'(k,q,k')}\right)} \cdot f'\left(k, q, k'\right)^{f'(k,q,k')}\right),$$

$$f_j(k, q, k') = O\left((60kqk')^{45kqk'} \cdot 2^{O\left(\sqrt{kqk' \cdot \log(kqk')}\right)} \cdot (kqk')^{kqk'}\right).$$

With a similar proof for the case that $q = \infty$, we have:

$$f'_j(k, k') = O\left(f\left(k, k'\right) \cdot 2^{O\left(\sqrt{f'(k,k') \cdot \log f'(k,k')}\right)} \cdot f'\left(k, k'\right)^{f'(k,k')}\right).$$

From Lemma 28, $f'(k, k') = O(kk')$ and $f(k, k') = O((15kk')^{45kk'})$, then

$$f'_j(k, k') = O\left((15kk')^{45kk'} \cdot 2^{O\left(\sqrt{kk' \cdot \log(kk')}\right)} \cdot (kk')^{kk'}\right).$$

□

Remarks and Structural Properties

Having shown the algorithm we can, now, provide the proof of Claim 1 which is: "For any partition function f compatible with a nice decomposition of some set A, the partition function *maxf* is also compatible".

Proof. Let $(D, \mathcal{X})$ be a nice decomposition of A with width not bigger than k'. Since f is compatible with $(D, \mathcal{X})$ we have that there are functions F_f and H_f such that for any partition $\mathcal{P}$ of A:

$$f_{A_v}(\mathcal{P}) = F_f(f_{A_u}(\mathcal{P} \cap A_u), \mathcal{P} \cap X_v, A_v \setminus A_u), \text{ and}$$
$$f_{A_v}(\mathcal{P}) = H_f(f_{A_u}(\mathcal{P} \cap A_u), f_{A_w}(\mathcal{P} \cap A_w), \mathcal{P} \cap X_v).$$

To show that *maxf* is compatible with $(D, \mathcal{X})$ we have to show that there are function F_{maxf} and H_{maxf} that play the same role as F_f and H_f.

Clearly, if $\mathcal{A} = \{A_1, A_2\}$ is a bipartition of A then $maxf(\mathcal{A}) = f(\mathcal{A})$, since $\{f(\{A_1, A_2\}) = f(\{A_2, A_1\})$. Hence, for any bipartition $\mathcal{P}$ of A, the function F_{maxf} takes the same value as F_f and the function H_{maxf} takes the same value as H_f.

To show how to compute F_{maxf}, during the processing of an introduce node $v \in D$ with child u with $A_v \setminus A_u = \{a\}$, consider the step "update of labels of vertex(s) and edge(s)" of procedure *IntroduceNode* when applied to a characteristic $C_u = ((T_u, r_u, \sigma_u), \ell_u, K_u, \text{dist}_u, out_u, branch_u, father_u) \in \text{FSC}_{\text{k,q}}(u)$ by subdividing an edge $f = \{v_{top}, v_{bottom}\}$.

Let $(T_u^\circ, r_u^\circ, \sigma_u^\circ)$ be a partitioning tree of A_u such that $C_u = \text{Char}((T_u, r_u, \sigma_u), X_u)$. Let f_u° be the representative of f_u in T_u°. Then, $(T_v^\circ, r_v^\circ, \sigma_v^\circ)$ is the partitioning tree for A_v obtained from $(T_u^\circ, r_u^\circ, \sigma_u^\circ)$ by subdividing the edge $f_u^\circ = \{v_{top}^\circ, v_{bottom}^\circ\}$ one time, creating a vertex v_{att}, and adding v_{leaf} as neighbor of v_{att}, make $\sigma_v^\circ(v_{leaf}) = a$ and $r_v^\circ = r_u^\circ$.

For each edge e of T_v let $\mathcal{T}_e$ ($\mathcal{T}_e^\circ$) be the partition of X_v (A_v) that it defines. Similarly, for each vertex t of T_v let $\mathcal{T}_t$ ($\mathcal{T}_t^\circ$) be the partition of X_v (A_v) that it defines.

Since an edge of T_v defines a bipartition of A_v, we can update the labels of edges in T_v using F_f, i.e. we apply the instruction "$\ell_v(e) \leftarrow F_f(\ell_u(e), \mathcal{T}_e, a)$" to edges of T_v. After this instruction, $\ell_v(e) = F_f(\ell_u(e), \mathcal{T}_e, a) = F_{maxf}(\ell_u(e), \mathcal{T}_e, a) = maxf(\mathcal{T}_e^\circ)$ for all edges of T_v.

For each internal vertex t of T_v, let E_t be the set of edges incident to t in T_v and let E_t° be the set of edges incident to t in T_v°. Assume, by induction, that $\ell_u(t) = maxf(\mathcal{T}_t^\circ \cap A_u)$.

From the fact that T_v° is a partitioning tree of A_v, we have that $\mathcal{T}_t^\circ = \{A_1, A_2, \ldots, A_{|E_t^\circ|}\}$ and, from the definition of *maxf*, we have $maxf(\mathcal{T}_t^\circ) = \max_{i=1}^{|E_t^\circ|} f(A_i, A_v \setminus A_i) = \max_{e \in E_t^\circ} \ell_v(e)$. Therefore, $F_{maxf}(maxf_{A_u}(t), \mathcal{T}_t, a) = F_{maxf}(\ell_u(t), \mathcal{T}_t, a)$ and $F_{maxf}(\ell_u(t), \mathcal{T}_t, a) = \max\{\ell_u(t), \max_{e \in E_t} \ell_v(e)\}$. Since the degree of t in T_v is bounded by k', F_{maxf} can be computed in constant time.

To show how to compute H_{maxf}, consider the step "merging the paths" of procedure *JoinNode* when applied to characteristics $C_u = ((T_u, r_u, \sigma_u), \ell_u, K_u, \text{dist}_u, out_u, branch_u, father_u) \in \text{FSC}_{\text{k,q}}(u)$ and $C_w = ((T_w, r_w, \sigma_w), \ell_w, K_w, \text{dist}_w, out_w, branch_w, father_w) \in \text{FSC}_{\text{k,q}}(w)$ during the computation of a join node $v \in D$ with children u and w. Let $(T_v^\circ, r_v^\circ, \sigma_v^\circ)$ be the partitioning tree for A_v obtained from the "merging" of $(T_u^\circ, r_u^\circ, \sigma_u^\circ)$, a partitioning tree for A_u with $C_u = \text{Char}((T_u^\circ, r_u^\circ, \sigma_u^\circ), X_u)$, and $(T_w^\circ, r_w^\circ, \sigma_w^\circ)$ with $C_w = \text{Char}((T_w^\circ, r_w^\circ, \sigma_w^\circ), X_w)$, a partitioning tree for A_w.

For each edge e of T_v let $\mathcal{T}_e$ ($\mathcal{T}_e^\circ$) be the partition of X_v (A_v) that it defines. Similarly, for each vertex t of T_v let $\mathcal{T}_t$ ($\mathcal{T}_t^\circ$) be the partition of X_v (A_v) that it defines.

Assume by induction that for any edge e of T_u (T_w) we have $\ell_u(e) = maxf(\mathcal{T}_e^\circ \cap A_u)$ ($\ell_w(e) = maxf(\mathcal{T}_e^\circ \cap A_w)$) and that for any vertex t of T_u (T_w) we have $\ell_u(t) = maxf(\mathcal{T}_t^\circ \cap A_u)$ ($\ell_w(t) = maxf(\mathcal{T}_t^\circ \cap A_w)$).

Let $P_u(x, y)$ be a path of C_u that is merged with a path $P_w(x, y)$ of C_w obtaining $P_v(x, y)$. Let $P'_u(x, y)$ and $P'_w(x, y)$ be the extensions used in the merging. We set $\ell_v(e) = H_f(\ell_u(e), \ell_v(e), \mathcal{T}_e)$ to any edge of $P_v(x, y)$. From the fact that e defines a bipartition of A_v, we have $\ell_v(e) = maxf(\mathcal{T}_e^\circ)$.

For each internal vertex t of $P_v(x,y)$, let E_{t_u} (E_{t_w}) be the set of edges incident to t in T_u (T_w) and let $E^\circ_{t_u}$ ($E^\circ_{t_w}$) be the set of edges incident to t in T°_u (T°_w).

Then, for each vertex t in $P_v(x,y)$ we set $\ell_v(t) = H_{maxf}(\ell_u(t), \ell_w(t), \mathcal{T}_t) = \max\{\ell_u(t), \ell_w(t), \max_{e\in E_t} \ell_v(e)\}$. From the induction hypothesis, $\max\{\ell_u(t), \ell_w(t), \max_{e\in E_t} \ell_v(e)\} = \max\{\max_{e\in E^\circ_{t_u}} maxf(\mathcal{T}_e \cap A_u), \max_{e\in E^\circ_{t_w}} maxf(\mathcal{T}_e \cap A_w), \max_{e\in E_t} \ell_v(e)\} = maxf(\mathcal{T}^\circ_t)$.

Since the degree of t in T_v is bounded by k', H_{maxf} can be computed in constant time. □

Lastly, we need to show how to take into account the "structural" properties of different width. Hence, it is necessary to guarantee that characteristics belonging to $\mathrm{FSC}_{\mathrm{k,q}}(v)$ of a node v in the nice decomposition correspond to partitioning trees that satisfy these structural properties. For example, partitioning trees for the branch width are such that every internal vertex has degree three, therefore for each node v of the nice decomposition $\mathrm{FSC}_{\mathrm{k,q}}(v)$ must contain only characteristics of partitioning trees for A_v which have every internal vertex with degree three. We show how to modify the algorithm to take into account the structural properties for each width mentioned in Section 4.1. The procedure *StartingNode* is modified to compute only the characteristics that respects the "structural" properties of the width being computed. We show which changes to *IntroduceNode* procedure and *JoinNode* procedure are necessary in order to achieve this. Let $G = (V, E)$ be a graph and $(D, \mathcal{X})$ a nice decomposition of $A = E$ given to the algorithm as input, in the case of carving width and cut width, $(D, \mathcal{X})$ is a nice decomposition of $A = V$.

Tree width and Path width: no changes are made to the procedures.

Special tree width: in the *IntroduceNode* at after step "update of $\mathrm{FSC}_{\mathrm{k,q}}(v)$", let $a = (x,y)$ be the element of $A_v \setminus A_u$ mapped by v_{leaf} and let X (Y) be the set of all leaves of T_v such that the edges of A_v they map have x (y) as one of its endpoints. Then, if the minimum spanning tree of T_v containing all vertices in X is not a caterpillar then the algorithm does not put C_v into $\mathrm{FSC}_{\mathrm{k,q}}(v)$ or if the minimum spanning tree of T_v containing all vertices in Y is not a caterpillar then the algorithm does not put C_v into $\mathrm{FSC}_{\mathrm{k,q}}(v)$. Procedure *JoinNode* remains unchanged.

Branch width, Linear width, Carving width and Cut width: we do not allow the Case 1 in the step "update of T_u into T_v" in the *IntroduceNode* procedure. That is, we do not allow a leaf mapping a to be added as neighbor of an internal vertex of T_u. In the *JoinNode* procedure during the step "merging the paths", we do not allow internal vertices of P_v, the result of merging P_u with P_w, to have a pair of vertices as its originators. In other words, any vertex of P_u must be "merged" with an edge of P_w and any vertex of P_w must be merged with an edge of P_u.

Time Complexity

The complexity of the algorithm to decide if a set A has q-branched Φ-width not bigger than k is given by the amount of time needed to compute a full set of characteristics for each node in the nice decomposition times the number of nodes in the nice decomposition. Let $(D, \mathcal{X})$ be the nice decomposition of A given as input to the algorithm. From the definition of a nice decomposition, we have that $|V(D)| = O(|A|)$. Let k' be equal to $\max_{X\in\mathcal{X}} |X|$. Hence, from Theorems 34, 36 and 38 the algorithm has time complexity bounded by

$$\max\{f_i(k,q,k'), f_f(k,q,k'), f_j(k,q,k')\} \cdot O(|A|).$$

Since the functions $f_i(k, q, k')$, $f_f(k, q, k')$ and $f_j(k, q, k')$ do not depend on $|A|$, if k, q and k' are given constants the algorithm has complexity bounded by $O(|A|)$. Therefore, it is a linear time algorithm when k, q and k' are fixed and it is a linear FPT-algorithm where the parameters are k, q and k'.

4.6 Conclusion

In this chapter, we use a generalization of width parameters of graphs, the partition functions and partitioning trees, to design a unified FPT algorithm to decide if the q-branched tree width, special tree width, branch width, linear width, cut width and carving width of graphs are not bigger than an integer k.

Unfortunately, the algorithm presented only solves the decision problem. That is, it can be used to decide if the q-branched tree width, special tree width, branch width, linear width and cut width of a graph is not bigger than an integer k, but it does not compute the respective decomposition.

In [BK96], Bodlaender and Kloks propose an algorithm that decides if the tree width of a graph is at most a given integer k and, if it is the case, it constructs a tree decomposition with this width. This algorithm, which in part inspired our algorithm, also makes use of the notion of "characteristic" of a tree decomposition and proceeds to compute these "characteristics" by a dynamic programming approach from a given tree decomposition of the graph. They also propose a second algorithm that constructs the tree decomposition from "characteristics" computed through the first algorithm.

In a current work, our algorithm was made constructive by following the same techniques used in their second algorithm, the one which constructs the tree decomposition for the input graph.

The algorithm we proposed in this chapter can compute several graph width measures, while not being restricted to only the aforementioned width measures. For example, since the rank width of graphs can be defined in terms of partition functions and partitioning trees [AMNT09], we wonder whether using this formalization, is it possible to design a FPT-algorithm computing the rank width.

We finish this part of the thesis with one question: are there other parameters of graphs that can be defined in terms of partition functions? We are specially interested in answering this question for directed graph decompositions, such as the Process Decomposition defined in Chapter 3 or the directed decompositions mentioned in Chapter 2 such as the directed tree decomposition, the Kelly decomposition or DAG decomposition. An answer to this question could be the first step into generalizing the results in this chapter to directed graph decompositions.

Part II

Turn-By-Turn Pursuit-Evasion Games

Chapter 5

Turn-by-Turn Pursuit-Evasion Games

In this part of the thesis, we focus on turn-by-turn pursuit-evasion games. One major difference between these games and the ones from the first part of the thesis is that the two players play alternately. That is, first one player plays while the other waits, then the other player plays while the first one waits.

In this chapter, we briefly survey some famous turn-by-turn pursuit-evasion games such as the *Cops and Robbers*, the *Eternal Dominating Set*, the *Eternal Vertex Cover* and the *Angel Problem*.

Along with the *Surveillance game* defined in the next chapter, these games can all be described with the framework proposed in Chapter 7. As a consequence, the results of Chapter 7 are valid for any of them.

5.1 Cops and Robbers

We start by describing the *Cops and Robbers* game due to its similarity with the *Graph Searching* games of the first part of this thesis.

The *Cops and Robbers* game can be roughly described as the *Node Search* game where both players play alternately, and the robber can move at most one single edge during its turn instead any path free of cops. As with the *Graph Searching* games, the *Cops and Robbers* game has several variations depending on the behaviors of the cops and of the robber. In this section, we explore some variations of the *Cops and Robbers* game. We start by defining the classical version of this game which was proposed by Nowakowski and Winkler [NW83] and, independently, by Quilliot [Qui83].

The classical version of the *Cops and Robbers* game has two players, the cop and the robber, playing on a graph. This is a turn-by-turn game with the first turn belonging to the cop. The game starts with the cop choosing a vertex of the graph to occupy followed by the robber who also chooses a vertex of the graph to occupy. Then, turn-by-turn, each player can move along an edge of the graph. The cop wins if, at any point during the game, it is able to occupy the same vertex as the robber. The robber wins if it can evade the cop indefinitely.

Nowakowski, Winkler and Quilliot were interested in a characterization of *cop-win* graphs, that is, graphs that the cop can win regardless of the moves of the robber. It is

easy to see that trees are cop-win graphs. Let T be any tree where the game is played. At each step, let v be the vertex occupied by the cop. Then, the cop moves to the neighbor of v in the minimum path from v to the robber. On the other hand, the cop cannot win against the robber in cycles of size at least four, since the robber can always move in the opposite direction of the cop, hence keeping its distance to the cop.

In the following, we reproduce the characterization of cop-win graphs given by Nowakowski and Winkler. Given a graph $G = (V, E)$, a vertex $v \in V$ is *irreducible* if there is a vertex $u \in V$, $v \neq u \in V$, such that $N[v] \subseteq N[u]$. In other words, the neighborhood of u covers the neighborhood of v. A graph G is said to be *dismantable* if there is an ordering of the vertices of G, $(v_1, \ldots, v_n)$, such that v_i is irreducible in $G[\{v_i, \ldots, v_n\}]$. In Figure 5.1, we show an example of dismantable graph.

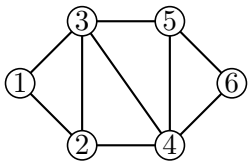


Figure 5.1: Example of a dismantable graph, with vertices numbered according to the ordering.

Theorem 39 ([NW83])**.** *A finite graph is cop-win if and only if it is dismantable.*

Nowakowski and Winkler also considered the problem of characterizing infinite graphs that are cop-win, giving a full characterization of such graphs by extending the notion of a dismantable graph to the infinite case. It seems unintuitive that a graph can be infinite connected and still be cop-win. However, it is easy to see that every infinite graphs with an universal vertex is cop-win. By occupying an universal vertex, the cop can ensure that it can capture the robber in its next turn. Moreover, there are also graphs that are cop-win, infinite and have no universal vertex. One example of such a graph can be seen in Figure 5.2. This graph is obtained by taking a path of size i, for each $i \in \mathbb{N}$, and joining one of its extremities with a central vertex v. A simple strategy for the cop is to start by placing itself at vertex v and then, at each step, moving towards the end of the path where the robber lies. Since each path has a finite size, the cop can capture the robber in a finite number of turns. In [AF84] the first polynomial time algorithm to decide if a graph is cop-win was described. This algorithm, essentially, computes an ordering of G that proves that G is dismantable. The complexity of this algorithm comes from a theorem in [AF84] that states that a graph G is cop-win if, and only if, by sequentially deleting (in any order) irreducible vertices, the result is a single vertex.

A *Cops and Robbers* game, where there is more than one cop trying to capture the robber, was first proposed by Aigner and Frome [AF84]. The minimum number of cops necessary to ensure the capture of the robber in a graph G is the cop number, denoted by $\mathrm{cn}(G)$. Clearly, for any graph G, $\mathrm{cn}(G)$ is at most the minimum domination number of G, since by placing one cop at each vertex of a dominating set the cops can capture the robber in their next move. On the other hand, if a graph G has minimum degree k and its girth is at least five, then $\mathrm{cn}(G) \geq k$ [AF84]. One natural question about this game is how big can be the cop number of a graph compared to the number of its vertices. Meyniel, in a personal communication with Frankl [Fra87] in 1987, stated the following conjecture:

Conjecture 1. For every graph G of order n, $\mathrm{cn}(G) = O(\sqrt{n})$ and, for every $n \in \mathbb{N}^*$, there is a graph G of order n such that $\mathrm{cn}(G) = \Omega(\sqrt{n})$.

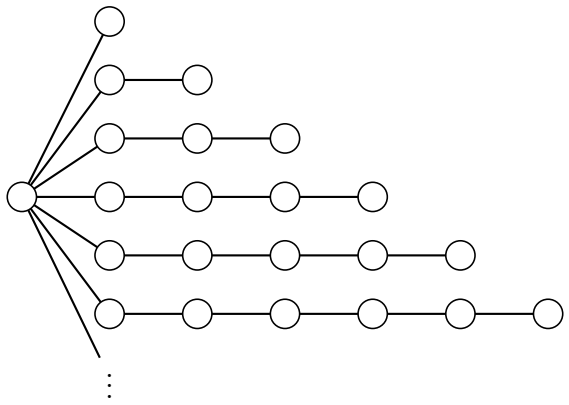


Figure 5.2: Example of cop-win graph that has no universal vertex.

This conjecture is arguably the most important open problem in the topic of *Cops and Robbers*. For a long time, the best known bound for the cop number of a graph with n vertices, given by Frankl [Fra87], was $O(n\frac{\log\log n}{\log n})$. This bound was further improved by Chinifooroshan [Chi08] to $O(\frac{n}{\log n})$. Currently, the best upper bound for the cop number of a general graph G of order n was independently proved by Lu and Peng [LP12], Frieze *et al.*[FKL12] and Scott and Sudakov [SS11].

Theorem 40 ([LP12, FKL12, SS11])**.** *For a any graph G with n vertices:*

$$\mathrm{cn}(G) = O\left(\frac{n}{2^{(1-o(1))\sqrt{\log_2 n}}}\right)$$

Despite the improvement, this newer bound and Meyniel's conjecture are still far from each other. Some further evidence pointing towards the truthfulness of this conjecture is due to Bollobás *et al.* [BKL13]. They prove that for sparse random graphs the cop-number has order of magnitude $n^{\frac{1}{2}+o(1)}$ with high probability.

Another important question is, given a graph G of order n, how small can be $\mathrm{cn}(G)$ compared to n. In other words, is it possible that, for every graph G with n vertices, we have $\mathrm{cn}(G) = o(\sqrt{n})$? The answer to this question is no. There are graphs G of order n with minimum degree k and girth at least 6 such that $k > \sqrt{2n}/2$ and $\mathrm{cn}(G) \geq k > \sqrt{2n}/2$. Then, from the fact that $\mathrm{cn}(G) \geq k$ [AF84], we have that $\mathrm{cn}(G) = \Omega(\sqrt{n})$. For more information about bounds and Meyniel's conjecture for the *Cops and Robbers* game, see [BB12].

Since a full characterization for graphs with cop number $k > 1$ is unknown, it is interesting to study the complexity of computing the cop number of a graph. Let $\mathrm{cn}_s(G)$ be the minimum number of cops that are necessary to ensure the capture of a robber that can move through at most s edges during its turn, but it is unable to pass through a vertex that is occupied by a cop. It was shown by Fomin *et al.* [FGK$^+$10] that, for all $s \geq 1$, it is NP-hard to compute $\mathrm{cn}_s(G)$ for a general graph G. Moreover, for a fixed value k and a general graph G of order n, deciding if $\mathrm{cn}(G) \leq k$, is W[2]-hard [FGK$^+$10]. It means that there are little chances of finding an algorithm with time complexity $O(f(k)n^{O(1)})$, where $f(k)$ is a function depending only on the parameter k.

Another complexity result due to Goldstein and Reingold [GR95] is that, assuming the initial positioning of the cops and the robber are given, that is, their positions are already defined before the game starts, then computing $\mathrm{cn}(G)$ is EXPTIME-complete. A more recent result due to Mamino [Mam13] shows that deciding if k cops can capture a robber

in a graph G is PSPACE-hard, even when both players can choose their starting positions. That is, deciding if $\mathrm{cn}(G) \leq k$, for a general graph G and integer k, is PSPACE-complete.

Due to the hardness of computing $\mathrm{cn}(G)$ for a general graph G and the difficulty of proving (or disproving) Meyniel's conjecture, a natural step is to study the cop number of graphs with certain properties. As mentioned before, trees are examples of cop-win graphs and graphs with an induced cycle of size at least four are not cop-win. In Table 5.1 we show some of the known results about computing the cop number of some graph classes.

Table 5.1: Complexity of computing, or the value of, $\mathrm{cn}(G)$, when G is a graph of order n belonging to the indicated class. In this table, $\mathrm{cn}_{\geq s}(G)$ denotes, for all $s' \geq s$, $\mathrm{cn}_{s'}(G)$ and $\mathrm{cn}_{1,2}(G)$ denotes both $\mathrm{cn}(G)$ and $\mathrm{cn}_2(G)$. The line with random graphs denotes binomial random graphs with probability p and the result essentially holds almost surely when $pn > 1$.

Graph Class	**Value/Difficulty**	**Reference**
Trees	$\mathrm{cn}(G) = 1$	[AF84]
Outerplanar	$\mathrm{cn}(G) \leq 2$	[Cla02]
Planar	$\mathrm{cn}(G) \leq 3$	[AF84]
Bounded Genus (g)	$\mathrm{cn}(G) \leq (3g/2) + 3$	[Sch01]
Series Parallel	$\mathrm{cn}(G) \leq 2$	[The08]
d-dimensional Grids	$\mathrm{cn}(G) \leq d$	[BPS10]
Geometric	$\mathrm{cn}(G) \leq 9$	[BDFM12]
Random	$O(n^{\frac{1}{2}-o(1)})$	[BKL13]
Chordal	polynomial time ($\mathrm{cn}(G)$)	[Qui86]
Interval	polynomial for $\mathrm{cn}_{\geq 1}(G)$	[FGK$^+$10]
Split	NP-hard for $\mathrm{cn}_{\geq 2}(G)$	[FGK$^+$10]
Bounded Clique-width	polynomial for $\mathrm{cn}_{1,2}(G)$	[FGK$^+$10]
k-Chordal ($k \geq 3$)	$\mathrm{cn}(G) \leq k - 1$	[KLNS12]

There are several variations of the *Cops and Robbers* games depending on the rules of the game and rules of capture. As mentioned before, one variation of the game allows the robber to move through more than one edge per turn.

In [HM06], a version of the game where cops must capture several robbers was studied. If each robber can be captured as soon as a cop occupies its current vertex, then all robbers can be captured with $\mathrm{cn}(G)$ cops. The cops simply need to capture each robber sequentially, while ignoring the other robbers. Hence, in this version, robbers must be captured simultaneously, that is, the cops only wins the game when all vertices that are occupied by robbers are also occupied by cops. They reduced the problem of deciding if k cops can capture l robbers in a directed graph D to the the problem of deciding if a directed graph D^*, obtained from D, is cop-win. However D^* might have an exponential number of vertices compared to D.

In [CCNV11], it was introduced a version of the game where only one cop tries to capture a robber on a graph with both moving at, possibly, different speeds. Let $\mathrm{CW}(s, s')$ be the set of graphs such that one cop moving at most s' edges per turn can capture one robber moving at most s edges per turn, regardless of the robbers moves. They show that $\mathrm{CW}(2, 1)$ is exactly the class of dually chordal[1] graphs. Moreover, for all $s \geq 3$ and $s' \geq 3$, they show that $\mathrm{CW}(s', 1) = \mathrm{CW}(s, 1)$. This essentially shows that if a cop can capture a fast robber, then it can capture a faster robber. They also study a version of the game

[1] A graph G is dually chordal if, and only if, there is a spanning tree T of G, such that every maximal clique of G induces a subtree in T.

where the robber is invisible to the cop, but it becomes visible to the cop periodically after each k rounds. They tried to characterize cop-win graphs for this version of the *Cops and Robbers* game for each value of k, by generalizing the dismantling order given in [NW83]. Unfortunately, this characterization is not tight. That is, every k-dismantable graph is cop-win when the robber is visible every k rounds, but the contrary is not necessarily true.

In [FGL12], Fomin *et al.* studied a version of the game where each cop has a fuel constraint. The fuel is an integer determining the amount of edges a cop can move through. The amount of fuel each cop has is the same at the beginning of the game. When a cop runs out of fuel, it cannot move any more, but it can still capture the robber, if the robber moves to the vertex it is occupying. They showed that deciding if k cops can capture the robber in this version of the game is PSPACE-complete, if the amount of fuel is at least two. The amount of fuel each cop has is just one of the possible restrictions that can be attributed to the cops. Other possible restrictions are the cost and the time of capture. The cost is the maximum amount of edges traversed by all cops and the time is the maximum amount of rounds the game can last. Fomin *et al.* studied the hardness of *Cops and Robbers* games, when cops are constrained to capture the robber with a maximum cost or in a maximum amount of turns. Deciding if k cops can capture a robber in a graph when constrained to a maximum fixed cost s, is NP-complete [FGP12]. If the cost is part of the input, but is still bounded by a polynomial on the size of the instance, this problem is PSPACE-complete [FGP12].

The first hardness result concerning the capture of a robber with constrained time was due to Bonato *et al.* in [BGHK09]. They showed that it is NP-complete to decide if k cops can capture a robber, when the maximum number of rounds is constrained to a constant. If the time is part of the input, but is a value bounded by a polynomial on the size of the instance, this problem is PSPACE-complete [FGP12].

For more knowledge about *Cops and Robbers* games see the book "The Game of Cops and Robbers on Graphs" from A. Bonato and R. Nowakowski [BN11].

5.2 Eternal Dominating Sets and Vertex Cover

In this section, we explore the *Eternal Dominating Sets* and *Eternal Vertex Cover*. In these games, the pursuer wins by protecting vertices, in the case of *Eternal Dominating Sets*, or edges, in the case of *Eternal Vertex Cover*, from an intrusion of the evader. The evader, on the other hand, wants to enter the area protected by the pursuer while avoiding its guards.

More precisely, in the pursuit-evasion game known as *Eternal Domination*, or *Eternal Security*, we have two players, the army and the rioter, that play turn-by-turn on a graph, $G = (V, E)$, with the army playing in the first turn. The army starts by choosing a dominating set, S_1, of the graph and placing one guard at each vertex of this dominating set. The rioter plays by choosing a single vertex, $v_1 \notin S_1$, of the graph. Then, at each subsequent round $i > 1$, the army must move one of the guards at a vertex u_i in $N[v_{i-1}]$ to v_{i-1} and the rioter chooses a new vertex v_i. In other words, after the initial placement of the guards, the rioter chooses a vertex v of G that is not occupied by a guard and the army must put one of its guards stationed at a neighbor of v at the vertex v. For each $i > 1$, let S_i be the set of vertices occupied by guards after the move of the army during round i, that is, $S_i = (S_{i-1} \setminus \{u_i\}) \cup \{v_{i-1}\}$. The rioter wins the game during round i, if

at the end of round i the set S_i is not dominating, while the army wins if the rioter never wins.

This game was first considered in [BCG+04]. Let $\sigma_1(G)$ be the minimum number of guards that are necessary for the army to win regardless of the sequence chosen by the rioter in the Eternal Domination game. Burger *et al.* show that $\sigma_1(G)$ is at least as big as the independence number of G and that it is at most as big as the clique cover number of G. There is a simple explanation for both of these results. Since the army must win for any sequence chosen by the rioter, then the rioter simply keeps choosing vertices in a maximum independent set of G. For the second one, let $S = \{C_1, C_2, \ldots, C_k\}$ be the set of cliques that covers G. Then by placing one guard at a vertex of each C_i, we have that this guard is able to "defend" against attacks on all vertices of this C_i. Burger *et al.* also provide formulas for the value of $\sigma_1(G)$ when G is a path, a cycle, a complete multipartite graph, hexagonal graph.

In [GHH05], Goddard *et al.* define a simple generalization of the game the m-Eternal Domination. In the m-Eternal Domination, the army can move all its guards independently along one edge of the graph with the restriction that at least one guard must be on the vertex chosen by the rioter after the movement. Let $\sigma_m(G)$ be the number of guards necessary to guarantee the victory of the army in the m-Eternal Domination game. The value of $\sigma_m(G)$ when G is complete, bipartite complete, a path or a cycle can be found in [GHH05]. They also show that the following holds for any graph G: $\sigma_m(G) \leq \gamma_2(G)$ and $\sigma_m(G) \leq \beta(G)$, where $\gamma_2(G)$ is the 2-domination number[2] and $\beta(G)$ is the independence number of G. Note that in [GK08] it was shown that there are graphs such that $\binom{\beta(G)}{2} \leq \sigma_1(G)$, hence proving that the gap between $\sigma_1(G)$ and $\sigma_m(G)$ can be high.

One question that remains open is the complexity of computing $\sigma_1(G)$ and $\sigma_m(G)$ for a general graph G.

Another game closely related to the m-Eternal Domination game is the *m-Eternal Vertex Cover* game. This game plays similarly to the m-Eternal Domination. The differences are as follows. The army, instead of choosing an initial dominating set, chooses a vertex cover. The rioter, instead of choosing vertices of the graph, chooses an edge of the graph, forcing the army to move one of its guards through the chosen edge during its turn. The rioter wins the game, if at the end of some round the set of vertices occupied by guards is not a covering of the edges. This game was first defined in [KM09]. Let $\alpha_m(G)$ be the minimum number of guards necessary for the army to win, regardless of the sequence of edges chosen by the rioter, in the m-Eternal Vertex Cover game.

A result relating m-Eternal Vertex Cover and m-Eternal Domination can be found in [KM11]. Klostermeyer and Mynhardt showed that $\alpha_m(G) \geq \sigma_m(G)$ for all G with minimum degree at least two. In [FGG+10], the problem of computing $\alpha_m(G)$ was shown to be NP-hard.

Fomin *et al.* propose a 2-approximation algorithm with complexity $O(\sqrt{n}m)$, where n and m are the number of vertices and edges of the input graph respectively. It was also proposed in [FGG+10] a FPT-algorithm to decide whether $\alpha_m(G) \leq k$, with complexity $O(2^{O(k^2)} + nm)$ where the parameter is k.

5.3 The Angel Problem

The last pursuit-evasion game we approach, in this chapter, is the *Angel problem*.

[2]Given a graph G, a set $S \subseteq V(G)$ is said to be k-dominating if, for all $v \in V \setminus S$, $|N(v) \cap S| \geq k$. The k-domination number is the minimum cardinality of a k-dominating set in G.

In this game, the two players are the devil and the angel. Each player is associated with an integer that defines its *power*. The game starts with the angel occupying a given vertex of the graph. Then, they play turn-by-turn starting with the devil. The devil plays by *eating* some vertices of the graph that are not occupied by the angel during its turn. The maximum amount of vertices the devil can eat on its turn is given by its power. During its turn, the angel plays by moving along edges of the graph and it must move through at least one edge. Similarly, the maximum amount of edges the angel can traverse on its turn is defined by its power. The angel wins the game if it is able to perpetually avoid being on an eaten vertex at the end of its turn.

Conway proposed a first version of this problem in [Con98]. In this version, the game is played on an infinite diagonal-grid[3] with both the angel and the devil having power one and. In other words, we can imagine the game being played on an infinite chessboard, where the moves of the angel are the moves of the king and the devil can mark one square of the chessboard at each turn. The objective is to know whether an angel with sufficient power can survive indefinitely. One first result shows that an angel of power one loses against any devil [Con98].

A first result of survivability for the angel was shown by Bollobás and Leader [BL06] and, independently, by Kutz [Kut05]. They show that an angel with sufficient power wins against a devil with power one in a tree-dimensional diagonal-grid. While Bollobás and Leader proof is not optimal with regards to the power of the angel, their survivability result also holds for a version of the *Angel problem* where the devil can eat more than one vertex per turn, but it is restricted to eat vertices that are sufficiently far away from the angel. On the other hand, Kutz's proof shows that an angel with power 13 is always able to survive against a devil with power one. Unfortunately, both results fail to hold for the game when it is played in a two-dimensional diagonal-grid.

The question if an angel with sufficient power is able to survive against a devil with power one in a diagonal grid remained open up to 2007, when, independently, Máthé [Má07], Kloster [Klo07] and Bowditch [Bow07] showed that an angel of sufficient power can survive. Máthé and Kloster show that an angel of power two can survive against a devil of power one while Bowditch only shows that an angel of power four can survive against a devil of power one. Máthé and Bowdith, prove that an angel of power two can survive by showing that the angel can survive in a special variant of the game. Unfortunately, it is not obvious how one can use the strategies for the angel in the variants of the game defined by Máthé and Bowdith to construct a winning strategy for the angel in the original game. On the other hand, Kloster was able to construct a winning strategy for the angel. This strategy is mainly based on the fact that, if the angel chooses a direction and move as fast as possible in this direction, then a devil with power one is not able to eat enough vertices in order to build a *trap* for the angel. Therefore, the angel is able to evade such traps by updating its intended direction of movement.

More information about the *Angel problem* can be found in [Kut04].

5.4 Objectives

In the next chapter, we aim at studying another turn-by-turn pursuit evasion game, the *Surveillance game*. This game is played by two players: the observer and the surfer. While the surfer plays like an angel with power one, the observer plays like the devil of

[3] An infinite diagonal grid is obtained through an infinite grid by adding edges between all vertices of a same face.

a power defined in the beginning of the game. The game, however, starts with the surfer placed on an "eaten" vertex. The objective of the surfer is to leave the "eaten" area, while the observer tries to avoid this happening. We aim at investigating the cost associated with enforcing the observer to keep the area marked connected.

Turn-by-turn games presented in this chapter, along with the *Surveillance game*, all share some similarities. Obviously these games are all played by two players in a turn-by-turn manner. Moreover, both players play by moving tokens along the edges of the graph or by adding/removing tokens on the vertices of the graph. In Chapter 7, we propose a framework that can be used to describe all aforementioned turn-by-turn games. This framework allows us to define fractional versions of these games. In other words, instead of having a whole token on a vertex, for example a cop, we might have fractions of cops spread through several vertices. The same being true for tokens that are controlled by each of the players. That is, both players might move/add/remove, depending on the game, fractions of tokens. We aim at investigating a method to decide, for any game that can be described with this framework, if the player playing the role of pursuer has a winning strategy. Then, we try to answer some natural questions that arise with this framework such as what is the relationship between a fractional version of a turn-by-turn game and its integral version.

Chapter 6

Surveillance Game

This chapter is dedicated to the study of yet another turn-by-turn pursuit-evasion game, the *Surveillance game*. The *Surveillance game* is a two-player game which involves one player moving a mobile agent, called *surfer*, along the edges of a graph, while a second player, called *observer*, marks the vertices of the graph. The surfer wins if it manages to reach an unmarked vertex. The observer wins otherwise, i.e., if it manages to mark all nodes before the surfer "escapes". This game was introduced by Fomin *et al.* in [FGJM+12] in order to model the problem of prefetching web-pages.

In the *connected Surveillance game*, the vertices marked by the observer must induce a connected component. One of the main goals of this chapter is to answer a question regarding the cost of connectivity which was introduced in [FGJM+12]. That is, we are interested to know how big can be the gap between the number of marks needed by the observer in these two versions of the game.

This chapter is divided as follows. The definition of the *Surveillance game* along with a brief overview on related results can be found in Section 6.1. Section 6.2 is dedicated to answering how much is the cost of connectivity. Then, in Section 6.3, we introduce another variant of this game, the *Online Surveillance game*, which is a restriction of the connected variant where the observer discovers progressively the graph the game is played. This online variant has to main motivations, the first one is that it has a closer relationship with the prefetching web-pages problem and the second one is that it might also help in understanding the cost of connectivity between the connected variant and the "classical" one. In Section 6.4, we conclude this chapter with a discussion about the results found in it.

6.1 The Surveillance Game

We start this section by formally defining the *Surveillance game* [FGJM+12]. Let $G = (V, E)$ be an undirected simple n-node graph, $v_0 \in V$, and $k \in \mathbb{N}^*$. Initially, the surfer stands at v_0, which is marked, and all other nodes are not marked. Then, turn-by-turn, the observer first marks k unmarked vertices and then the surfer may move to a neighbor of its current position. Once a node has been marked, it remains marked until the end of the game. The surfer wins if, at some step, it reaches an unmarked vertex, otherwise the observer wins. Note that the game lasts at most n/k turns. When the game is played on a directed graph, the surfer has to move following the direction of the arcs. A *k-strategy*

for the observer from v_0, or simply a *k-strategy from* v_0, is a function $\sigma : V \times 2^V \to 2^V$. This function receives as input a vertex of the graph, representing the current position of the surfer and a set of vertices M of G, representing the set of marked vertices, returning a set M' of vertices of $V \setminus M$, representing the set of vertices that the observer should mark during its turn, such that $|M'| \leq k$. We emphasis that σ depends implicitly on the graph G, i.e., it is based on the full knowledge of G. A k-strategy from v_0 is *winning*, if it allows the observer to win whatever be the sequence of moves of the surfer, when it starts in v_0. The *surveillance number* of a graph G with initial node v_0, denoted by $\mathrm{sn}(G, v_0)$, is the smallest k such that there exists a winning k-strategy starting from v_0.

Let us define some notations that are used in this chapter. Let Δ be the maximum degree of the nodes in G and, for any $v \in V$, let $N(v)$ be the set of neighbors of v. More generally, the neighborhood $N(F)$ of a set $F \subseteq V$ is the subset of vertices of $V \setminus F$ which have a neighbor in F. Moreover, we define the closed neighborhood of a set F as $N[F] = N(F) \cup F$.

As an example, let us consider the following *basic strategy*: let $\sigma_{\mathcal{B}}$ be the strategy defined by $\sigma_{\mathcal{B}}(v, M) = N(v) \setminus M$ for any $M \subseteq V$, $v_0 \in M$, and $v \in M$. Intuitively, the basic strategy $\sigma_{\mathcal{B}}$ asks the observer to mark all unmarked neighbors of the current position of the surfer. It is straightforward, and it was already shown in [FGJM+12], that $\sigma_{\mathcal{B}}$ is a winning strategy for any $v_0 \in V$ and it easily implies that $\mathrm{sn}(G, v_0) \leq \max\{|N(v_0)|, \Delta - 1\}$.

Web-Page Prefetching, Connected and Online Variants

The *Surveillance game* has been introduced to model the web-page prefetching problem. This problem can be stated as follows. A web-surfer is following the hyper-links in the digraph of the web. The web-browser aims at downloading the web-pages before the web-surfer accesses it. The number of web-pages that the browser may download before the web-surfer accesses another web-page is limited, due to bandwidth constraints. Therefore, designing efficient strategies for the *Surveillance game* would allow to preserve bandwidth while, at the same time, avoiding the web-surfer to wait the download of the web-page he wants to access.

By the nature of the web-page prefetching problem, in particular, because of the huge size of the web digraph, it is not realistic to assume that a strategy may mark any node of the network, even nodes that are "far" from the current position of the surfer. For this reason, [FGJM+12] defines the *connected* variant of the *Surveillance game*. A strategy σ is said *connected* if $\sigma(v, M) \cup M$ induces a connected subgraph of G for any M, $v_0 \in M \subseteq V(G)$. Note that the basic strategy $\sigma_{\mathcal{B}}$ is connected. The *connected surveillance number* of a graph G with initial node v_0, denoted by $\mathrm{csn}(G, v_0)$, is the smallest k such that there exists a winning connected k-strategy starting from v_0. By definition, $\mathrm{csn}(G, v_0) \geq \mathrm{sn}(G, v_0)$ for any graph G and $v_0 \in V(G)$. In [FGJM+12], it is shown that there are graphs G and $v_0 \in V(G)$ such that $\mathrm{csn}(G, v_0) = \mathrm{sn}(G, v_0) + 1$. Only the trivial upper bound $\mathrm{csn}(G, v_0) \leq \Delta\, \mathrm{sn}(G, v_0)$ is known, and a natural question is how big the gap between $\mathrm{csn}(G, v_0)$ and $\mathrm{sn}(G, v_0)$ may be [FGJM+12]. This chapter provides a partial answer to this question.

Still the *connected Surveillance game* seems unrealistic since the web-browser cannot be asked to have the full knowledge of the web digraph. For this reason, we define the *Online Surveillance game*. In this game, the observer discovers the graph while marking its nodes. That is, initially, the observer only knows the starting node v_0 and its neighbors. After the observer has marked the subset M of nodes, it knows M and the

vertices that have a neighbor in M. The next set of vertices to be marked depends only on this knowledge and on the position of the surfer, i.e., the nodes at distance at least two from M are unknown. In other words, an *online strategy* is based on the current position of the surfer, the set of already marked nodes and on the subgraph H of the marked nodes and their neighbors (a more formal definition is postponed to Section 6.3). By definition, the set of nodes marked by such a strategy, at each step, must be known, i.e., adjacent to an already marked vertex. Therefore, an online strategy is also connected. We are interested in the competitive ratio of winning online strategies. The competitive ratio $\rho(\mathcal{S})$ of a winning online strategy $\mathcal{S}$ is defined as $\rho(\mathcal{S}) = \max_{G,v_0 \in V(G)} \frac{\mathcal{S}(G,v_0)}{\mathrm{sn}(G,v_0)}$, where $\mathcal{S}(G, v_0)$ denotes the maximum number of vertices marked by $\mathcal{S}$ in G at each turn, when the surfer starts in v_0. Note that, because any online winning strategy $\mathcal{S}$ is connected, $\mathrm{csn}(G, v_0) \leq \rho(\mathcal{S})\,\mathrm{sn}(G, v_0)$, for any graph G and $v_0 \in V(G)$.

Related Work

The *Surveillance game* has mainly been studied in the computational complexity point of view. It is shown that the problem of computing the surveillance number is NP-hard in split graphs [FGJM$^+$12]. Moreover, deciding whether the surveillance number is at most 2 is NP-hard in chordal graphs, and deciding whether the surveillance number is at most 4 is PSPACE-complete. Polynomial-time algorithms that compute the surveillance number in trees and interval graphs are proposed in [FGJM$^+$12]. All previous results also hold for the connected surveillance number. Finally, it is shown that, for any graph G and $v_0 \in V(G)$, $\max \left\lceil \frac{|N[S]|-1}{|S|} \right\rceil \leq \mathrm{sn}(G, v_0) \leq \mathrm{csn}(G, v_0)$, where the maximum is taken over every subset $S \subseteq V(G)$ inducing a connected subgraph with $v_0 \in S$. Moreover, both previous inequalities turn into an equality in the case of trees. In [FGJM$^+$12], Fomin *et al.* ask for an example where these inequalities are strict.

In the literature, there are mainly three types of approaches in order to solve the web-page prefetching problem: server based hints prefetching [AEFP98, AZN99, Mog96], local prefetching [WLZC12] and proxy based prefetching [FCLJ99]. In local prefetching, the client has no aid from the server when deciding which documents to prefetch. In the server based hints prefetching, the server can aid the client to decide which pages to prefetch. Lastly, in the proxy based prefetching, a proxy that connects its clients with the server decides which pages to prefetch. Moreover, some studies consider that the prefetching mechanism has perfect knowledge of the web-surfer's behavior [PM96, KLM97]. In these studies, the objective is to minimize the waiting time of the web-surfer with a given bandwidth, by designing good prediction strategies for which pages to prefetch.

In the context of prefetching web-pages, the *Surveillance game* is a model to study a local prefetching scheme to guarantee that a web-surfer never has to wait a web-page to be downloaded, whilst minimizing the bandwidth necessary to achieve such a goal. For this, the web-surfer takes the role of the *surfer*, the browser takes the role of the *observer* and the graph the game is played is the digraph of the web.

Our Results

In this chapter, we study both the connected and online variants of the *Surveillance game*. First, we try to evaluate the gap between non-connected and connected surveillance number of graphs. We give a new upper bound, independent from the maximum degree, for the ratio $\mathrm{csn} \,/\, \mathrm{sn}$. More precisely, we show that, for any n-node graph G and any $v_0 \in V(G)$, $\mathrm{csn}(G, v_0) \leq \sqrt{\mathrm{sn}(G, v_0) \cdot n}$. Then, we describe a family of graphs G such

that $\mathrm{csn}(G, v_0) = \mathrm{sn}(G, v_0)+2$. Note that, contrary to the simple example that shows that connected and not connected surveillance number may differ by one, a larger difference seems much more difficult to obtain.

As mentioned, the online variant of the *Surveillance game* is a more constrained version of the connected game. We prove that any online strategy has competitive ratio at least $\Omega(\Delta)$. More formally, we describe a family of trees with constant surveillance number such that, for any online winning strategy, there is a step when the strategy has to mark at least $\Delta/4$ vertices in order to win. Unfortunately, this shows that the best (up to constant ratio) online strategy is the one that simply marks the out-neighborhood of the current position of the surfer.

Unless otherwise stated, all graphs in this chapter are undirected simple and connected. Note that, in undirected graphs, the out-neighborhood and in-neighborhood of a vertex coincide.

6.2 Cost of Connectivity

In this section, we investigate the cost of enforcing connectivity. We prove the first non-trivial upper bound for the ratio $\mathrm{csn}\,/\,\mathrm{sn}$. More precisely, we show that for any n-node graph G, $\mathrm{csn}(G, v_0) \leq \sqrt{\mathrm{sn}(G, v_0) \cdot n}$. Then, we improve the lower bound of [FGJM+12]. That is, we show a family of graphs where $\mathrm{csn}(G, v_0) > \mathrm{sn}(G, v_0)+1$. Finally, we disprove a conjecture in [FGJM+12].

Upper Bound for the Cost of Connectivity

The next result is the first non-trivial upper bound (independent from the degree) of the cost of the connectivity in the *Surveillance game*.

Theorem 41. *Let G be any connected n-node graph and $v_0 \in V(G)$, then*

$$\mathrm{csn}(G, v_0) \leq \sqrt{\mathrm{sn}(G, v_0) \cdot n}.$$

Proof. $\mathrm{sn}(G, v_0) = 1$ if and only if G is a path with v_0 as one of the extremities. In this case, $\mathrm{csn}(G, v_0) = \mathrm{sn}(G, v_0)$ and the result holds.

Assume that $k = \mathrm{sn}(G, v_0) > 1$ and that $n \geq 2$. We describe a connected strategy σ marking at most $\sqrt{kn}$ nodes per turn. Let $M^0 = \{v_0\}$ and let M^t be the set of vertices marked after $t \geq 1$ turns. Assume moreover that M^t induces a connected graph of G containing v_0. Finally, let v_t be the vertex occupied by the surfer after turn t. The set $\sigma(v_t, M^t)$ of nodes marked by the observer at step $t + 1$ is defined as follows. If $|V(G) \setminus M^t| \leq \sqrt{kn}$, then let $\sigma(v_t, M^t) = V(G) \setminus M^t$. Otherwise, let $H \subseteq V(G) \setminus M^t$ be such that $|H| = \sqrt{kn}$, $H \cup M^t$ induces a connected subgraph and $|H \cap N(v_t)|$ is maximum. Then, $\sigma(v_t, M^t) = H$, i.e., the strategy marks $\sqrt{kn}$ new nodes in a connected way and, moreover, it marks as many unmarked nodes as possible among the neighbors of v_t. In particular, if $|N(v_t) \setminus M^t| \leq \sqrt{kn}$, then all neighbors of v_t are marked after turn $t + 1$.

By definition, σ is connected and marks at most $\sqrt{kn}$ nodes per turn. We need to show σ is winning.

For purpose of contradiction, let us assume that the surfer wins against σ by following the path $P = (v_0, \ldots, v_t, v_{t+1})$. At its $t+1^{th}$ turn, the surfer moves from a marked vertex v_t to an unmarked vertex v_{t+1}.

Therefore, $n > t\sqrt{kn}$, otherwise the observer marking $\sqrt{kn}$ nodes at each turn would have already marked every vertex on the graph by the end of turn t. Moreover, by definition of sigma, $|N(v_t) \setminus M^t| > \sqrt{kn}$

Since, $\mathrm{sn}(G, v_0) = k$, let $\mathcal{S}$ be any k-winning (non necessarily connected) strategy for the observer. Assume that the observer follows $\mathcal{S}$ against the surfer following $P \setminus \{v_{t+1}\}$. Since, $\mathcal{S}$ is winning, all vertices of $N(v_t)$ must be marked after turn t, otherwise the surfer would win by moving to an unmarked neighbor of v_t. Therefore, since $\mathcal{S}$ can mark at most k vertices each turn, $|N(v_t)| \leq kt$.

Taking both inequalities, we have that $\sqrt{kn} < |N(v_t)| \leq kt$. Hence, $\sqrt{n} < t\sqrt{k}$. Since $n > t\sqrt{kn}$ and $\sqrt{n} < t\sqrt{k}$, we have that $t^2k < n < t^2k$, a contradiction. □

Lower Bound for the Cost of Connectivity

This subsection is devoted to proving the following theorem.

Theorem 42. *There exists a family of graphs G and $v_0 \in V(G)$ such that* $\mathrm{csn}(G, v_0) > \mathrm{sn}(G, v_0) + 1$.

We use the following result proved in [FGJM$^+$12]. For any graph $G = (V, E)$ and any vertex $v_0 \in V$, a k-strategy for G with initial vertex v_0 is winning if and only if it is winning against a surfer that is constrained to follow induced paths on G. In other words, the walk of the surfer is constrained to be an induced path.

In this section, by *adding a path $P = (v_1, \cdots, v_r)$ between two vertices u and v of G*, we mean that the induced path P is added as an induced subgraph of G and the edges $\{u, v_1\}$ and $\{v_r, v\}$ are added.

Let $x \geq 4$, α, β and γ be four strictly positive integers satisfying the following inequations.

$$\max\{\beta, \frac{\beta}{2} + \gamma + 1\} < \alpha < \min\{\beta + \gamma + 1, 2\gamma + 2\} \tag{6.1}$$

$$\beta < 2\gamma + 2 \tag{6.2}$$

$$\alpha + \beta + 2\gamma + 12 \leq 3x \tag{6.3}$$

$$\frac{4}{5}(\alpha + \beta + \gamma) + 10 < x \tag{6.4}$$

$$73 + \beta + 2\gamma \leq 2\alpha \tag{6.5}$$

For instance, $x = 250$, $\alpha = 146$, $\beta = 73$, $\gamma = 73$ are values that satisfy all the above inequalities.

For proving the main theorem in this section we mainly rely in the family of graphs built in the following the procedure described below.

Let $\mathcal{G} = (V, E)$ be a graph with 10 isolated vertices $\{v_0, w_0, w_1, w_2, w'_0, w'_1, w'_2, s_0, s_1, s_2\}$. Then, for all $i \in \{0, 1, 2\}$ do the following:

1. $4x - 9$ vertices of degree one are added and made adjacent to s_i;
2. $3x - 2$ vertices of degree one are added and made adjacent to w_i, respectively $3x - 2$ neighbors of degree one are added to w'_i;
3. two disjoint paths $A^i = (a^i_1, \ldots, a^i_\alpha)$ and $A'^i = (a'^i_1, \ldots, a'^i_\alpha)$ are added between v_0 and s_i;

4. a path $B^i = (b^i_1, \ldots, b^i_\beta)$ is added between v_0 and w_i, and a path $B'^i = (b'^i_1, \ldots, b'^i_\beta)$ is added between v_0 and w'_i;

5. for any $j \in \{i, i+1 \mod 3\}$ a path $C^{i,j} = (c^{i,j}_1, \ldots, c^{i,j}_\gamma)$ is added between s_j and w_i, and a path $C'^{i,j} = (c'^{i,j}_1, \ldots, c'^{i,j}_\gamma)$ is added between s_j and w'_i;

6. for any $1 \leq j \leq \alpha$, $3x-1$ vertices of degree one are added and made adjacent to a^i_j, respectively $3x-1$ neighbors of degree one are added to a'^i_j;

7. for any $1 \leq j \leq \beta$, $3x-1$ vertices of degree one are added and made adjacent to b^i_j, respectively $3x-1$ neighbors of degree one are added to b'^i_j;

8. for any $1 \leq j \leq \gamma$, $\ell \in \{i, i+1 \mod 3\}$, $3x-1$ vertices of degree one are added and made adjacent to $c^{i,\ell}_j$, respectively $3x-1$ neighbors of degree one are added to $c'^{i,\ell}_j$.

The shape of $\mathcal{G}$ is depicted in Figure 6.1. $\mathcal{G}$ has $(30+18(\alpha+\beta)+36\gamma)x-29$ vertices. For any $i \in \{0,1,2\}$, the node s_i has $4x-3$ neighbors, v_0 has 12 neighbors, and any other non-leaf node has degree $3x+1$.

Claim 14. *If* $\max\{\beta, \frac{\beta}{2}+\gamma+1\} < \alpha < \min\{\beta+\gamma+1, 2\gamma+2\}$ *and* $\beta < 2\gamma+2$*, the unique (up to symmetries) minimum Steiner-tree for* $S = N[v_0] \cup \{s_0, s_1, s_2\}$ *in* $\mathcal{G}$ *has* $15+\alpha+\beta+2\gamma$ *vertices and consists of the vertices of the paths* A^0, B^1, $C^{1,1}$, $C^{1,2}$ *and the vertices in* $S \cup \{w_1\}$.

Proof. The subgraph induced by the vertices of the paths A^0, B^1, $C^{1,1}$, $C^{1,2}$ and the vertices in $S \cup \{w_1\}$ is a subtree spanning S and with $15+\alpha+\beta+2\gamma$ vertices. Let us enumerate all the possible (up to symmetries) Steiner-trees for S. Consider the subgraph induced by the vertices of:

- A^0, A^1, A^2 and S. The number of vertices in this subgraph is $3\alpha+13$.
- $A^0, A^1, C^{1,1}, C^{1,2}$ and $S \cup \{w_1\}$. The number of vertices in this subgraph is $2\alpha+2\gamma+15$.
- $A^0, A^1, B^1, C^{1,2}$ and $S \cup \{w_1\}$. The number of vertices in this subgraph is $2\alpha+\beta+\gamma+14$.
- $A^0, C^{0,0}, C^{0,1}, C^{2,0}, C^{2,2}$ and $S \cup \{w_0, w_2\}$. The number of vertices in this subgraph is $\alpha+4\gamma+17$.
- $B^0, B^1, C^{0,0}, C^{1,1}, C^{1,2}$ and $S \cup \{w_0, w_1\}$. The number of vertices in this subgraph is $2\beta+3\gamma+16$.
- $B^1, C^{1,1}, C^{1,2}, C^{2,2}, C^{2,0}$ and $S \cup \{w_1, w_2\}$. The number of vertices in this subgraph is $\beta+4\gamma+17$.

If the subgraph induced by the vertices of the paths A^0, B^1, $C^{1,1}$, $C^{1,2}$ and the vertices in $S \cup \{w_1\}$, is the unique (up to symmetries) minimum Steiner-tree for $S =$

$N[v_0] \cup \{s_0, s_1, s_2\}$ in G, then we get the following inequalities:

$$\begin{aligned}
\alpha &> \frac{\beta}{2} + \gamma + 1 \\
\alpha &> \beta \\
\alpha &> \gamma + 1 \\
\beta &< 2\gamma + 2 \\
\alpha &< \beta + \gamma + 1 \\
\alpha &< 2\gamma + 2.
\end{aligned}$$

Thus $\max\{\beta, \frac{\beta}{2} + \gamma + 1\} < \alpha < \min\{\beta + \gamma + 1, 2\gamma + 2\}$ and $\beta < 2\gamma + 2$. □

In Figure 6.1, the scheme of a minimum Steiner-tree for $S = N[v_0] \cup \{s_0, s_1, s_2\}$ is depicted with dashed lines.

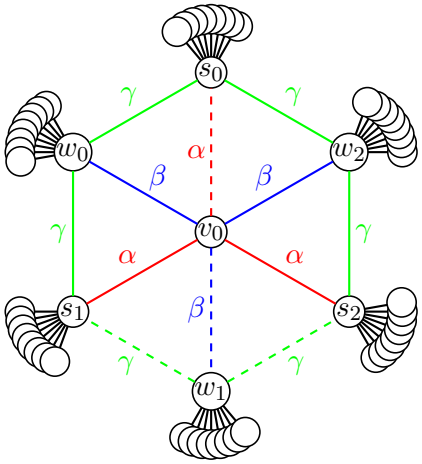


Figure 6.1: Graph Family Scheme. Here we show only one "layer" of the graph.

For any $i \in \{0, 1, 2\}$, let $\mathcal{A}_i = N[v_0] \cup N[A^i] \cup N[s_i]$ (resp., $\mathcal{A}'_i = N[v_0] \cup N[A'^i] \cup N[s_i]$). Note that $|\mathcal{A}_i| = |\mathcal{A}'_i| = (3\alpha + 4)x + 9$ and that the $\mathcal{A}_i$ and $\mathcal{A}_j$, $i \neq j$, pairwise intersect only in $N[v_0]$.

For any $i \in \{0, 1, 2\}$, let $\mathcal{B}_i = N[v_0] \cup N[B^i] \cup N[w_i] \cup N[C^{i,i}] \cup N[C^{i,i+1 \mod 3}] \cup N[s_i] \cup N[s_{i+1 \mod 3}]$ and $\mathcal{B}'_i$ is defined similarly. $|\mathcal{B}_i| = |\mathcal{B}'_i| = (3\beta + 6\gamma + 11)x + 5$. Finally, for any $i \in \{0, 1, 2\}$ and $j \in \{i, i + 1 \mod 3\}$, let $\mathcal{B}_{i,j} = N[v_0] \cup N[B^i] \cup N[w_i] \cup N[C^{i,j}] \cup N[s_j]$ and $\mathcal{B}'_{i,j} = N[v_0] \cup N[B'^i] \cup N[w'_i] \cup N[C'^{i,j}] \cup N[s_j]$.

Lemma 43. *For any $i \in \{0, 1, 2\}$ and $j \in \{i, i + 1 \mod 3\}$, during its first step, any winning $(3x + y)$-strategy for $\mathcal{G}$ must mark at least*

- *$x + 8 - y(\alpha + 1)$ nodes in $\mathcal{A}_i$ (resp., in $\mathcal{A}'_i$), and*
- *$x + 8 - y(\beta + \gamma + 2)$ nodes in $\mathcal{B}_{i,j}$ (resp., in $\mathcal{B}'_{i,j}$), and*
- *$2x + 4 - y(\beta + 2\gamma + 3)$ nodes in $\mathcal{B}_i$ (resp., in $\mathcal{B}'_i$).*

Proof. Let $\mathcal{S}$ be any winning $(3x + y)$-strategy and F be the set of nodes that $\mathcal{S}$ marks during its first step.

Let $M = F \cap \mathcal{A}_0$. The surfer goes to a_1^0. We may assume that $\mathcal{S}$ had marked it since the strategy fails otherwise. Now, the surfer first goes to s_0 through A^0 unless, at

some turn, its position has an unmarked neighbor. In the latter case, the surfer goes to this unmarked node and wins. During these $(\alpha+1)$ steps, the strategy $\mathcal{S}$ can mark at most $(\alpha+1)(3x+y)$ extra nodes in $\mathcal{A}_0$. Hence, in total, at most $|M|+(\alpha+1)(3x+y)$ nodes have been marked in $\mathcal{A}_0$ when the surfer is at s_0 and it is its turn. Because $\mathcal{S}$ is a winning strategy, all nodes in $\mathcal{A}_0$ must have been marked since otherwise the surfer would have won. Therefore, $|M|+(\alpha+1)(3x+y) \geq |\mathcal{A}_0 \setminus \{v_0\}| = (3\alpha+4)x+8$ and $|M| \geq x+8-y(\alpha+1)$.

The proof is similar for $\mathcal{B}_{i,j}$.

Now, let $M = F \cap \mathcal{B}_0$ and let $M' = F \cap (N[v_0] \cup N[B^0] \cup N[w_0]) \subseteq M$. The surfer goes to b_1^0. We may assume that $\mathcal{S}$ had marked it since the strategy fails otherwise. Now, the surfer first goes to w_0 through B^0 unless, at some turn, its position has an unmarked neighbor. In the latter case, the surfer goes to this unmarked node and wins. At the turn of the surfer when it is in w_0, the strategy has marked $|M|+(\beta+1)(3x+y)$ and all nodes in $N[v_0] \cup N[B^0] \cup N[w_0]$ must have been marked. Therefore, at most $|M|+(\beta+1)(3x+y)-(12+3(\beta+1)x) = |M|+y(\beta+1)-12$ nodes of $\mathcal{B}'_0 \setminus (N[v_0] \cup N[B^0] \cup N[w_0])$ are marked. Hence, w.l.o.g., there are at most $\left\lfloor \frac{|M|+y(\beta+1)-12}{2} \right\rfloor$ nodes that are marked in $(N[C^{0,0}] \cup N[s_0]) \setminus N[w_0]$. The surfer now goes from w_0 to s_0. During these steps, at most $(\gamma+1)(3x+y)$ new vertices are marked. Because $\mathcal{S}$ is a winning strategy, all nodes in $(N[C^{0,0}] \cup N[s_0]) \setminus N[w_0]$ must have been marked since otherwise the surfer would have won. Therefore,

$$\left\lfloor \frac{|M|+y(\beta+1)-12}{2} \right\rfloor + (\gamma+1)(3x+y) \geq |(N[C^{0,0}] \cup N[s_0]) \setminus N[w_0]| \geq 3\gamma x + 4x - 4.$$

Hence, $|M| \geq 2x+4-y(\beta+2\gamma+3)$. □

Lemma 44. $\mathrm{sn}(\mathcal{G}, v_0) = 3x$.

Proof. First, let us show that $\mathrm{sn}(\mathcal{G}, v_0) \leq 3x$. For this purpose, consider the following strategy. At the first step, the observer marks the 12 neighbors of v_0 and, for any $i \in \{0,1,2\}$, the observer marks $x-4$ one-degree neighbors of s_i.

Note that, all nodes in $N(v_0)$ have exactly $3x$ unmarked neighbors and any vertex has at most $3x+1$ unmarked neighbors. Now, the strategy simply consists in marking at each step the neighbors of the current position of the surfer. Indeed, it is easy to prove by induction on the number of steps that, each time that the surfer arrives at a new node, this node is marked and has at most $3x$ unmarked neighbors.

Now, let us prove that $\mathrm{sn}(\mathcal{G}, v_0) > 3x-1$. Let $\mathcal{S}$ be any $(3x-1)$-strategy and let F be the set of nodes that $\mathcal{S}$ marks during its first step. Clearly, $N(v_0) \subseteq F$ since otherwise the surfer wins after its first move. Moreover, because the sets $\mathcal{A}_i \setminus N[v_0]$ are pairwise disjoint, there must be $i \in \{0,1,2\}$, such that $|F \cap (\mathcal{A}_i \setminus N[v_0])| < x-4$. Hence, $|F \cap \mathcal{A}_i| < x+8$ for some i. However, by Lemma 43, any winning $(3x-1)$-strategy must mark at least $x+8+\alpha+1 > x+8$ nodes in each $\mathcal{A}_i$ during the first step. □

Lemma 45. $\mathrm{csn}(\mathcal{G}, v_0) > 3x+1$.

Proof. For purpose of contradiction, let us assume that there is a winning connected $3x+1$-strategy. Let F be the set of vertices marked by this strategy during the first step. Clearly, $N(v_0) \subseteq F$ and $|F| \leq 3x+1$.

For any $0 \leq i \leq 2$, let $f_i = |F \cap N[s_i]|$ and let $f_{min} = \min_i f_i$. Without loss of generality, $f_{min} = f_0$. We first show that $f_{min} > 3$.

By Lemma 43, for any $i \in \{0,1,2\}$, $|F \cap (\mathcal{A}_i \setminus N[v_0])| \geq x - 5 - \alpha$ and, for any $i \in \{0,2\}$, $|F \cap (\mathcal{B}_{i,0} \setminus N[v_0])| \geq x - 6 - (\beta + \gamma)$ and $|F \cap (\mathcal{B}'_{i,0} \setminus N[v_0])| \geq x - 6 - (\beta + \gamma)$. Therefore,

$$\begin{aligned} 3x + 1 &\geq |F \cap (\mathcal{A}_0 \cup \mathcal{A}'_0 \cup \mathcal{A}_1 \cup \mathcal{A}_2 \cup \mathcal{B}_{0,0} \cup \mathcal{B}_{2,0} \cup \mathcal{B}'_{0,0} \cup \mathcal{B}'_{2,0})| \\ &\geq 12 + 4(x - 5 - \alpha) + 4(x - 6 - (\beta + \gamma)) - 5|F \cap N[s_0]| \\ &\geq 8x - 4(\alpha + \beta + \gamma) - 32 - 5f_{min} \end{aligned}$$

Hence, $5f_{min} \geq 5x - 4(\alpha + \beta + \gamma) - 33$, and $f_{min} \geq x - \frac{4}{5}(\alpha + \beta + \gamma) - 7 > 3$ by the above inequality.

Therefore, by definition of f_{min}, $|F \cap N[s_i]| \geq 4$ for any $i \in \{0,1,2\}$. By connectivity of the strategy, $s_i \in F \cap N[s_i]$ for any $i \in \{0,1,2\}$. Hence, F must contain a subset of vertices inducing a subtree spanning $S = N[v_0] \cup \{s_0, s_1, s_2\}$. Let T be an inclusion-minimal subset of F that induces a subtree spanning S. By Claim 14, $|T| \geq \alpha+\beta+2\gamma+15$. Let $T' = T \setminus (N[v_0] \cup \bigcup_{0 \leq i \leq 2} N[s_i])$. Then, $|T'| \geq \alpha + \beta + 2\gamma - 4$. Moreover, because of the symmetries, we may assume w.l.o.g., that $T' \subseteq \bigcup_{0 \leq i \leq 2}(\mathcal{A}_i \cup \mathcal{B}_i)$.

By Lemma 43 and because $N(v_0) \subseteq F$, for any $0 \leq i \leq 2$, $|F \cap (\mathcal{A}'_i \cup \mathcal{B}'_{i+1 \mod 3})| \geq x + 8 - (\alpha + 1) + 2x + 4 - (\beta + 2\gamma + 3) - 12 = 3x - (\alpha + \beta + 2\gamma) - 4$. Hence, $|T'| + |F \cap (\mathcal{A}'_i \cup \mathcal{B}'_{i+1 \mod 3})| \geq 3x - 8$. Let $W_i = F \setminus (\mathcal{A}'_i \cup \mathcal{B}'_{i+1 \mod 3} \cup T')$. Since $|F| \leq 3x + 1$, it follows that $|W_i| \leq 9$.

Let $f_{max} = \max_i f_i$ and assume w.l.o.g. that $f_{max} = f_2$. Since $\sum_{0 \leq i \leq 2} f_i \leq |F \setminus T'|$, we get that $f_0 + f_1 \leq \left\lfloor \frac{2}{3}(5 + 3x - (\alpha + \beta + 2\gamma)) \right\rfloor$.

To conclude, $|F \cap \mathcal{B}'_0| = |N(v_0)| + f_0 + f_1 + |W_0| \leq 21 + \left\lfloor \frac{2}{3}(5 + 3x - (\alpha + \beta + 2\gamma)) \right\rfloor$. On the other hand, Lemma 43 implies that $|F \cap \mathcal{B}'_0| \geq 2x + 1 - (\beta + 2\gamma)$. Therefore, $22 + \frac{2}{3}(5 + 3x - (\alpha + \beta + 2\gamma) > 2x + 1 - (\beta + 2\gamma)$ and it follows $73 > 2\alpha - \beta - 2\gamma$. This contradicts the inequalities. □

Lemmas 44 and 45 are sufficient to prove Theorem 42. More precisely, it shows that there exist a family of graphs G and $v_0 \in V(G)$ such that $\mathrm{csn}(G, v_0) \geq \mathrm{sn}(G, v_0) + 2$. However, as shown in the next lemma, the family of graphs we described does not allow to increase more the lower bound on the cost of connectivity.

Lemma 46. $\mathrm{csn}(\mathcal{G}, v_0) \leq 3x + 2$.

Proof. Consider the following strategy. At the first step, the observer marks the 12 neighbors of v_0, all nodes of the paths A^0, B^1, $C^{1,1}$ and $C^{1,2}$, the vertices w_1, s_0, s_1 and s_2 and finally $Z = \lfloor (3x - \alpha - \beta - 2\gamma - 12)/3 \rfloor$ one-degree neighbors of each s_i. Note that $Z \geq 0$ by Equation 6.3.

Then, the strategy goes on as follows. Let $i \in \{0,1,2\}$. When the surfer arrives at some node a^i_j (resp., a'^i_j), $1 \leq j \leq \alpha$, the observer marks the at most $3x$ unmarked neighbors of a^i_j and marks at least 2 unmarked neighbors of s_i. When the surfer arrives at some node b^i_j (resp., b'^i_j), $1 \leq j \leq \beta$, or at w_i, the observer marks the at most $3x$ unmarked neighbors of this node and marks at least 1 unmarked neighbor of s_i and at least 1 unmarked neighbor of $s_{i+1 \mod 3}$. When the surfer arrives at some node $c^{i,\ell}_j$ (resp., $c'^{i,\ell}_j$), $1 \leq j \leq \gamma$, $\ell \in \{i, i+1 \mod 3\}$, the observer marks the at most $3x$ unmarked neighbors of $c^{i,\ell}_j$ and marks at least 2 unmarked neighbors of s_ℓ (if any) and, if

all neighbors of s_ℓ are already marked, the observer marks at least 2 unmarked neighbors of s_k where $\{k\} = \{i, i+1 \mod 3\} \setminus \{\ell\}$. Finally, when the surfer arrives at s_i, the observer marks $3x+2$ unmarked neighbors of it.

To prove the validity of this strategy, it is sufficient to show that the surfer will loose for the following three different trajectories. This is sufficient, because the surfer is only able to win when moving from s_0, s_1 or s_2 to one of its neighbors; and because $\alpha < 2\gamma$, i.e., the amount of steps it takes for the surfer to move from s_i to s_j, with $j \neq i$ is bigger than the amount of steps it takes it to move from v_0 to s_j. Meaning that, if the fugitive wins it wins the first time it moves out of one of these three vertices.

First, let us assume that the surfer goes from v_0 to s_i through A^i ($i \in \{0,1,2\}$). Clearly, at each step before reaching s_i, all neighbors of the current position of the surfer are marked. Now, when the surfer arrives at s_i, there are at least $2(\alpha+1)+Z$ neighbors of s_i that are already marked. To show that the observer wins, it is sufficient to note that

$$\begin{aligned}
|N(s_i)| - (2(\alpha+1)+Z) &= 4x - 3 - 2\alpha - 2 - \left\lfloor \frac{3x-\alpha-\beta-2\gamma-12}{3} \right\rfloor \\
&\leq 3x - 2\alpha - 5 + \frac{\alpha+\beta+2\gamma+12}{3} \\
&\leq 3x - 1 + \frac{\beta+2\gamma-5\alpha}{3} \\
&\leq 3x+2
\end{aligned}$$

because $2\alpha > \beta + 2\gamma + 1$.

Second, let us assume that the surfer goes from v_0 to s_i through B^i, w_i and $C^{i,i}$ ($i \in \{0,1,2\}$). When the surfer arrives at s_i, there are at least $\beta+1+2\gamma+Z$ neighbors of s_i that are already marked. To show that the observer wins, it is sufficient to note that

$$\begin{aligned}
|N(s_i)| - (\beta+1+2\gamma+Z) &= 4x - 4 - \beta - 2\gamma \\
&\quad - \left\lfloor \frac{3x-\alpha-\beta-2\gamma-12}{3} \right\rfloor \\
&\leq 3x - \beta - 4 - 2\gamma + \frac{\alpha+\beta+2\gamma+12}{3} \\
&\leq 3x + \frac{\alpha-2\beta-4\gamma}{3} \\
&\leq 3x+2
\end{aligned}$$

because $\alpha < \beta + \gamma + 1$.

Finally, let us assume that the surfer goes from s_i (all neighbors of which are already marked) to $s_{i+1 \mod 3}$ through $C^{i,i}$, w_i and $C^{i,i+1 \mod 3}$ ($i \in \{0,1,2\}$). When the surfer arrives at $s_{i+1 \mod 3}$, there are at least $4\gamma+2+Z$ neighbors of $s_{i+1 \mod 3}$ that are already

marked. To show that the observer wins, it is sufficient to note that

$$\begin{aligned}|N(s_{i+1 \mod 3})| - (4\gamma + 2 + Z) &= 4x - 3 - 4\gamma - 2 \\ &\quad - \left\lfloor \frac{3x - \alpha - \beta - 2\gamma - 12}{3} \right\rfloor \\ &\leq 3x - 5\gamma - 4 + \frac{\alpha + \beta + 2\gamma + 12}{3} \\ &\leq 3x - 1 + \frac{\alpha + \beta - 10\gamma}{3} \\ &\leq 3x + 2\end{aligned}$$

because $\beta < \alpha < 2\gamma + 1$. □

To conclude this section, we answer negatively a question asked in [FGJM$^+$12]. It is shown in [FGJM$^+$12] that in the case of trees both following inequalities turn into equalities. For any graph G and $v_0 \in V(G)$, $\max \left\lceil \frac{|N[S]|-1}{|S|} \right\rceil \leq \mathrm{sn}(G, v_0) \leq \mathrm{csn}(G, v_0)$ where the maximum is taken over every subset $S \subseteq V(G)$ inducing a connected subgraph with $v_0 \in S$. Moreover, the authors of [FGJM$^+$12] ask whether the first inequality may be strict when the G is not restricted to be a tree.

First, let us notice that an equality might give new way to attack the question of the cost of the connectivity. However, such an equality is unlikely to hold since it would imply that the problem of computing the surveillance number of a graph is in co-NP while this problem is known to be PSPACE-complete in DAGs [FGJM$^+$12]. We actually show that there are graphs where the inequality is strict.

Let us build a graph as follows. Starting from the vertex set $V = \{a, b, c, ab, ac, bc, s\}$ and $E = \{(s, a), (s, b), (s, c), (a, ab), (a, ac), (b, ab), (b, bc), (c, ac), (c, bc)\}$. Then, we add $\frac{11k-21-2x}{6}$ leaves to each vertex ab, ac and bc, moreover, add 3 leaves to each vertex a, b and c, and, finally, add x leaves to s. A scheme of this family can be found in Figure 6.2.

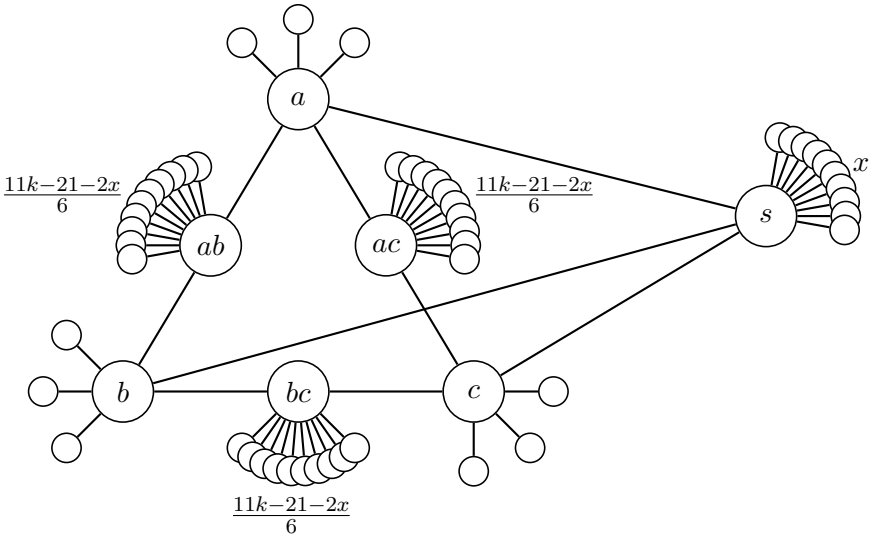


Figure 6.2: Scheme of the graph family described in the proof of Theorem 47.

We moreover assume that $k-5 \equiv 0 \pmod 2$, $k-x-3 \equiv 0 \pmod 3$, $11k-21-2x \equiv 0 \pmod 6$, $x \leq k-36$ and $k \geq 34$. For instance, $k = 105$ and $x = 42$ are possible values for k and x.

Let $\mathcal{G}$ be the graph obtained by the above construction and where parameters satisfy the above constraints.

Theorem 47. $\mathrm{sn}(\mathcal{G}, s) = k$ *and* $\max_{S \subseteq V(\mathcal{G})} \left\lceil \frac{|N[S]|-1}{|S|} \right\rceil < k$.

Proof. Throughout this proof, let $M \subseteq V$ denote the set of (currently) marked vertices in $\mathcal{G}$.

We show a strategy for the surfer that wins against an observer that can mark at most $k-1$ vertices per turn. Let

$$\begin{aligned} S_a &= (N[a] \cup N[ab] \cup N[ac]) \setminus \{s, a, b, c\}, \\ S_b &= (N[b] \cup N[ab] \cup N[bc]) \setminus \{s, a, b, c\}, \text{ and} \\ S_c &= (N[c] \cup N[bc] \cup N[ac]) \setminus \{s, a, b, c\}. \end{aligned}$$

In the first step and after the observer has used its marks, the surfer chooses to move to i where $i = \arg\min_{i=\{a,b,c\}} |S_i \cap M|$. Since the observer must mark the vertices in $N(s)$ (including a, b, c) we have that $|S_i \cap M| \leq \frac{2}{3}(k-1-x-3)$. Without loss of generality assume that $i = a$. In the second step, all neighbors of a must have been marked, otherwise the surfer wins by moving to an unmarked leaf of a. Let $S_{ab} = N[ab] \setminus \{a, b, ab\}$ and $S_{ac} = N[ac] \setminus \{a, c, ac\}$, therefore, after all marks are spent in the second step,

$$\min_{j=\{ab,ac\}} |S_j \cap M| \leq \frac{k-1-5+\frac{2}{3}(k-1-x-3)}{2}.$$

The surfer then chooses to move to $\arg\min_{i=\{ab,ac\}} |S_i \cap M|$, w.l.o.g. assume that it is the vertex ab. In the third step, the observer might use all its available marks onto the leaves of ab, hence, after spending all the marks,

$$|S_{ab} \cap M| \leq k-1+\frac{k-1-5+\frac{2}{3}(k-1-x-3)}{2} = \frac{11k-32-2x}{6}$$

which is less than $|S_{ab}|$, hence there is an unmarked leaf of ab that the surfer can reach.

We consider now a winning strategy for the observer that marks k vertices per step. At the first step, the observer marks all vertices in $N[s]$, with the remaining marks, $k-x-3$, being spread evenly among vertices in the sets $N[ab] \setminus \{a, b, ab\}$, $N[ac] \setminus \{a, c, ac\}$ and $N[bc] \setminus \{b, c, bc\}$. Hence, there are at least $\left\lfloor \frac{k-x-3}{3} \right\rfloor = \frac{k-x-3}{3}$ vertices marked in each of those sets. Without loss of generality assume that the surfer moves towards a. Then, the observer marks the vertices in $N(a)$ and, with the remaining marks, proceeds to distribute them evenly among the vertices of the sets $N(ab)$ and $N(ac)$. When the surfer is about to move there are at least $\left\lfloor \frac{k-5}{2} \right\rfloor + \frac{k-x-3}{3} = \frac{k-5}{2} + \frac{k-x-3}{3}$ vertices in $(N(ab) \setminus \{a, b\}) \cap M$ and in $(N(ac) \setminus \{a, c\}) \cap M$. Without loss of generality assume that the surfer moves towards ab. Then the observer uses all its available marks on the unmarked vertices in $N(ab) \setminus \{a, b\}$. Therefore, after all marks are spent, there are $k + \frac{k-5}{2} + \frac{x-3}{3}$ marked vertices in $N(ab) \setminus \{a, b\}$. It remains to show that $k + \frac{k-5}{2} + \frac{x-3}{3} \geq \frac{11k-21-2x}{6}$.

$$\begin{aligned} k + \frac{k-5}{2} + \frac{x-3}{3} &\geq \frac{6k}{6} + \frac{3k-15}{6} + \frac{2x-6}{6} - 2 = \frac{9k-21+2x}{6} - 2 \\ \frac{9k-21+2x}{6} - 2 &= \frac{9k-33+4x-2x}{6} \geq \frac{11k-21-2x}{6}. \end{aligned}$$

Now we show that for all connected sets S such that $s \in S$ we have that $\left\lceil \frac{|N[S]-1|}{|S|} \right\rceil < k$.

Claim 15. *For all connected sets S such that $s \in S$, then*

$$\left\lceil \frac{|N[S]-1|}{|S|} \right\rceil \leq k-1.$$

First we prove that if S contains a vertex $v \in V$ with degree 1, then $\left\lceil \frac{|N[S]-1|}{|S|} \right\rceil \leq \left\lceil \frac{|N[S\setminus\{v\}]-1|}{|S\setminus\{v\}|} \right\rceil$. Since S contains s and induces a connected subgraph, then $N(v) \subset S$ because $|N(v)| = 1$. Thus $N[S \setminus \{v\}]$ contains v and so $N[S \setminus \{v\}] = N[S]$.

In the rest of the proof, we consider sets S that do not contain a node with degree 1. Let $L_{ab} = N(ab) \setminus \{a, b\}$, $L_{ac} = N(ac) \setminus \{a, c\}$, and $L_{bc} = N(bc) \setminus \{b, c\}$. By the previous assumption, if a node $v \in L_{ab}$ is such that $v \in N[S]$, then all nodes in L_{ab} are in $N[S]$. By symmetry, we have a similar property for L_{ac} and L_{bc}. Note that $(N(s) \setminus \{a, b, c\}) \subset N[S]$ because $s \in S$ by definition.

We have four different cases:

1. Consider S such that $N[S] \cap (L_{ab} \cup L_{ac} \cup L_{bc}) = \emptyset$. We get that $|S| \geq 1$ and $N[S] \leq x + 16$. Thus $\left\lceil \frac{|N[S]-1|}{|S|} \right\rceil \leq x + 15 \leq k - 1$ because $x \leq k - 36$.

2. Consider S such that $N[S] \cap (L_{ac} \cup L_{bc}) = \emptyset$ and $L_{ab} \subset N[S]$. We get that $|S| \geq 3$ and $N[S] \leq x + 16 + \frac{11k-21-2x}{6}$. Thus $\left\lceil \frac{|N[S]-1|}{|S|} \right\rceil \leq \left\lceil \frac{11k+4x+69}{18} \right\rceil \leq k - 1$ because $x \leq k - 36$ and $k \geq 34$. The case $N[S] \cap (L_{ab} \cup L_{bc}) = \emptyset$ and $L_{ac} \subset N[S]$ is similar and the case $N[S] \cap (L_{ab} \cup L_{ac}) = \emptyset$ and $L_{bc} \subset N[S]$ is also similar.

3. Consider S such that $N[S] \cap L_{bc} = \emptyset$ and $L_{ab} \cup L_{ac} \subset N[S]$. We get that $|S| \geq 4$ and $N[S] \leq x + 16 + \frac{11k-21-2x}{3}$. Thus $\left\lceil \frac{|N[S]-1|}{|S|} \right\rceil \leq \left\lceil \frac{11k+x+24}{12} \right\rceil \leq k - 1$ because $x \leq k - 36$ and $k \geq 34$. The case $N[S] \cap L_{ac} = \emptyset$ and $L_{ab} \cup L_{bc} \subset N[S]$ is similar and the case $N[S] \cap L_{ab} = \emptyset$ and $L_{ac} \cup L_{bc} \subset N[S]$ is also similar.

4. Consider S such that $L_{ab} \cup L_{ac} \cup L_{bc} \subset N[S]$. We get that $|S| \geq 6$ and $N[S] \leq x + 16 + \frac{11k-21-2x}{2}$. Thus $\left\lceil \frac{|N[S]-1|}{|S|} \right\rceil \leq \left\lceil \frac{11k+9}{12} \right\rceil \leq k - 1$ because $k \geq 34$.

This concludes the proof of Claim 15 and, therefore, the proof of Theorem 47 because we have proved that $\mathrm{sn}(\mathcal{G}, s) = k$. □

6.3 Online Surveillance Number

In this section, we study the online variant of the *Surveillance game*. This variant is motivated by the web-page prefetching problem. In this variant, the observer does not know *a priori* the graph in which it is playing. That is, initially, the observer only knows v_0, its degree and the identifiers of its neighbors. Then, when a new node is marked, the observer discovers all its neighbors that are not yet marked. Note that the degree of a node is not known before it is marked.

Another property of an online strategy that must be defined is if the observer must use all its marks simultaneously or in sequence. Assume that the set M of nodes have been marked and it is the turn of the observer. Either it first chooses the k nodes that will be marked among the set $N(M) \setminus M$, the unmarked neighbors of the nodes that were already marked, and then the observer marks each of these k nodes and discover their unknown neighbors simultaneously; or the observer first chooses one node x in $N(M) \setminus M$, marks it and discovers its unmarked neighbors, then it chooses a new node to be marked

in $N(M \cup \{x\}) \setminus (M \cup \{x\})$ and so on until the observer finishes its turn after marking k nodes. We choose to consider the second model because it is less restricted, i.e., the observer has more power. Even in this case, our result is pessimistic since we show that the *basic strategy*, which marks the neighborhood of the current position of the surfer at each step, is the best one with respect to the competitive ratio.

Formal Definition of Online Strategy and Competitive Ratio

Now we are ready to formally define an online strategy. Let $k \geq 1$, let $G = (V, E)$ be a graph, $v_0 \in V$, and let $\mathcal{G}$ be the set of subgraphs of G.

Let $M \subseteq V$ be a subset of nodes inducing a connected subgraph containing v_0 in G. Let $G_M \in \mathcal{G}$ be the subgraph of G known by the observer when M is the set of marked nodes. That is, $G_M = (M \cup N(M), E_M)$ where $E_M = \{(u, v) \in E \mid u \in M\}$. For any $u, v \in N(M) \setminus M$, let us set $u \sim_M v$ if and only if $N(u) \cap M = N(v) \cap M$. Let $\mathcal{X}_M$ be the set of equivalent classes, called *modules*, of $N(M) \setminus M$ with respect to $\sim_M$. The intuition is that two nodes in the same module of $\mathcal{X}_M$ are known by the observer but cannot be distinguished. For instance, $\mathcal{X}_{\{v_0\}} = \{N(v_0)\}$ because initially all neighbors of v_0 look the same to the observer.

A *k-online strategy for the observer starting from* v_0 is a function $\sigma : \mathcal{G} \times V \times 2^V \times \{1, \ldots, k\} \to 2^V$ such that, for any subset $M \subseteq V$ of nodes inducing a connected subgraph containing v_0 in G, for any $v \in M$, and for any $1 \leq i \leq k$, we have that $\sigma(G_M, v, M, i) \in \mathcal{X}_M$. This means that, if M is the set of nodes already marked and v is the position of the surfer and it remains $k - i + 1$ nodes to be marked by the observer before the surfer moves, then the observer will mark one node in $\sigma(G_M, v, M, i)$.

More precisely, we say that the observer *follows* the k-online strategy σ if the game proceeds as follows. Let $M = M^0$ be the set of marked nodes just after the surfer has moved to $v \in M$. Initially, $M^0 = \{v_0\}$ and $v = v_0$. Then, the strategy proceeds sequentially in k steps for $i = 1$ to k. First, the observer marks an arbitrary node $x_1 \in \sigma(G_{M^0}, v, M^0, 1)$. Let $M^1 = M^0 \cup \{x_1\}$. Sequentially, after having marked $1 < i < k$ nodes at this turn, the observer marks one arbitrary node $x_{i+1} \in \sigma(G_{M^i}, v, M^i, i + 1)$ and $M^{i+1} = M^i \cup \{x_{i+1}\}$. When the observer has marked k nodes, that is after choosing $x_k \in \sigma(G_{M^{k-1}}, v, M^{k-1}, k)$, it is the turn of the surfer, then it may move to a node adjacent to its current position and a new turn for the observer starts. Note that because we are interested in the worst case for the observer, each marked node $x_i \in \sigma(G_{M^{i-1}}, v, M^{i-1}, i)$ is chosen by an adversary.

The *online surveillance number* of a graph G with initial node v_0, denoted by $\mathrm{on}(G, v_0)$, is the smallest k such that there exists a winning k-online strategy starting from v_0. In other words, there is a winning k-online strategy σ starting from v_0 such that an observer following σ wins whatever be the trajectory of the surfer and the choices done by the adversary. Note that, since we consider the worst scenario for the observer, we may assume that the surfer and the adversary have full knowledge of G.

Online Surveillance Number vs Surveillance Number

Theorem 48. *There exists a infinite family of rooted trees such that, for any T with root $v_0 \in V(T)$ in this family,* $\mathrm{sn}(T, v_0) = 2$ *and* $\mathrm{on}(T, v_0) = \Omega(\Delta)$ *where Δ is the maximum degree of T.*

Proof. We first define the family $(T_k)_{k \geq 4}$ of rooted trees as follows.

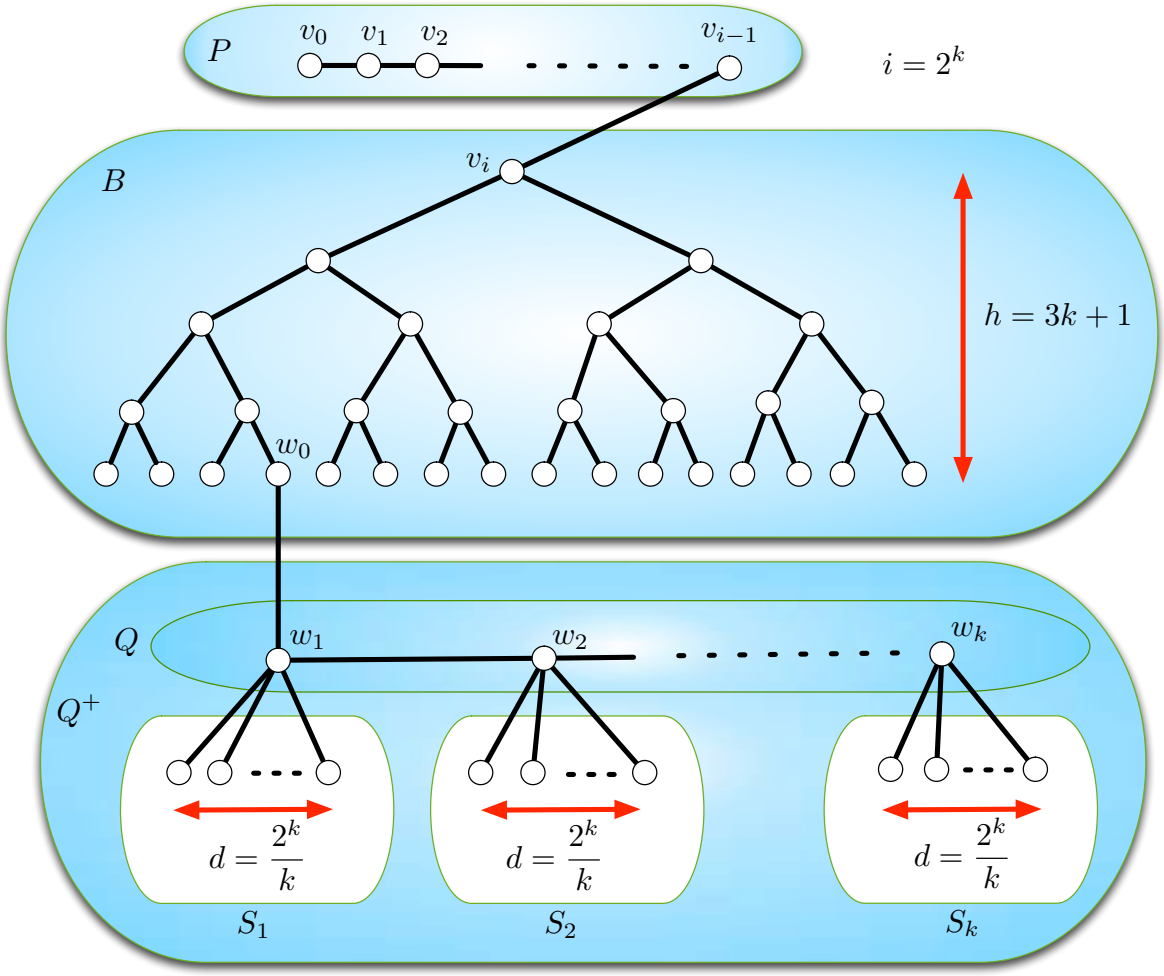


Figure 6.3: Tree T_k described in the proof of Theorem 48.

Let $k \geq 4$ be a power of two and let $i = 2^k$ and $d = \frac{2^k}{k}$.

Let us consider a path $P = (v_0, v_1, \ldots, v_{i-1})$ with i nodes Let B be a complete binary tree of height $h = 3k + 1$ and rooted at some vertex v_i, i.e., B has $2^{h+1} - 1$ vertices. Let w_0 be any leaf of B. Finally, let $Q = (w_1, \ldots, w_k)$ be a path on k nodes. Note that, P, B and Q depend on k.

The tree T_k is obtained from P, B and Q by adding an edge between v_{i-1} and v_i, an edge between w_0 and w_1. Finally, for any $1 \leq j \leq k$, let us add an independent set, S_j, with d vertices and an edge between each vertex of S_j and w_j (i.e., each node in S_j is a leaf in the resulting tree T_k). T_k is then rooted in v_0.

Let Q^+ denote the union of vertices of Q and $\bigcup_{j=1}^{k} S_j$. The maximum degree Δ of T_k is reached by any node w_j, $1 \leq j < k$, and $\Delta = d + 2 = \frac{2^k}{k} + 2$.

We first show that $\mathrm{sn}(T_k, v_0) = 2$. Clearly, $\mathrm{sn}(T_k, v_0) > 1$. Let us consider the following (offline) strategy for the observer. At each turn $j \leq i$, the surfer marks the vertex v_j and one unmarked vertex of Q^+ that is closest to the surfer. Note that the observer is allowed to mark nodes in Q^+ because, in an offline strategy, the observer knows the whole tree. Just after turn i, the surfer must occupy a node of $P \cup \{v_i\}$. Moreover, it cannot have reached an unmarked vertex so far since all nodes of $P \cup \{v_i\}$ have been marked before the surfer can access them.

For each turn $j > i$ and while the surfer does not occupy a node in $Q^+ \cup \{w_0\}$, the

observer marks the neighbors of the current position of the surfer if they are not already marked. While the surfer remains on the nodes of B or P, this strategy clearly requires to mark at most 2 nodes per turn since B is a binary tree.

Finally, if the surfer occupies a node in $Q^+ \cup \{w_0\}$, the observer marks two unmarked nodes of Q^+ that are closest to the surfer. It only remains to prove that the surfer cannot reach an unmarked node in Q^+. When the surfer reaches w_0, this node must be marked by the previous strategy. Moreover, by the strategy of the first i turns, the i nodes of Q^+ that are closest to w_0 have been marked. Hence, for any $1 \leq j \leq k$, when the surfer reaches w_j, at least the $i + 2j$ nodes of Q^+ that are closest to w_0 are marked. Since $|\bigcup_{1\leq p\leq j} N[w_p] \cap Q^+| \leq j(d+1)+1$ and because $i = dk$, we get that $i + 2j = dk + 2j \geq dj + j + 1 \geq |\bigcup_{1\leq p\leq j} N[w_p] \cap Q^+|$ and therefore, the surfer never reaches a node with an unmarked neighbor.

Hence, $\mathrm{sn}(T_k, v_0) = 2$.

Now it remains to show that $\mathrm{on}(T_k, v_0) = \Omega(\Delta)$. Let γ be any online strategy for T_k and marking at most $\frac{d}{4} = \frac{2^{k-2}}{k}$ nodes per turn. We show that γ fails.

For this purpose, we model the fact that the observer does not know the graph by "building" the tree during the game. More precisely, each time the observer marks a node v, then the adversary may add new nodes adjacent to v or decide that v is a leaf. Of course, the adversary must satisfy the constraint that eventually the graph is T_k. Initially, the observer only knows v_0 that has one neighbor v_1. Now, for any $1 \leq j < i$, when the observer marks the node v_j of P, then the adversary "adds" a new node v_{j+1} adjacent to v_j, i.e., the observer discovers its single unmarked neighbor v_{j+1}. Now, let v be any node of B. Recall that h is the height of B. When the observer marks v, there are three cases to be considered: if v is at distance at most $h-1$ from v_i, then the adversary adds two new nodes adjacent to v; if v is at distance h from v_i and not all nodes of B have been marked then the adversary decides that v is a leaf; finally, if all nodes of B have been marked (v is the last marked node of B, i.e., B is a complete binary tree of height h), the adversary decides that $v = w_0$ and add one new neighbor w_1 adjacent to it. Note that we can ensure that the last node of B to be marked is at distance h of v_i by connectivity of any online strategy.

Now, let consider the following execution of the game. During the first i steps, the surfer goes from v_0 to v_i. Just after the surfer arrives in v_i, the observer has marked at most $(di)/4$ nodes and all nodes of $P \cup \{v_i\}$ must be marked since otherwise the surfer would have won. Therefore, at most $i(d/4-1)+1 = 2^{2k-2}/k - 2^k + 1$ nodes of B are marked when it is the turn of the surfer at v_i. Since B has $2^{h+1} - 1 = 2^{3k+2} - 1$ nodes, at least one node of B is not marked.

From v_i, the surfer always goes toward w_0. Note that the observer may guess this strategy but it does not know where is w_0 while all nodes of B have not been marked.

Then let $0 \leq t \leq h$ and let $v'_t \in V(B)$ be the position of the surfer at step $i + t$ and B^t the subtree of B rooted at v'_t. Note that, at step i, $v'_0 = v_i$ and $B^0 = B$. Let B^t_l and B^t_r be the subtrees of B rooted at the children of v'_t. W.l.o.g., let us assume that the number of marked nodes in B^t_l is at most the number of marked nodes in B^t_r, when it is the turn of the surfer standing at v'_t. Then, the surfer moves to the root of B^t_l. That is, v'_{t+1} is the child of v_t whose subtree contains the minimum number of marked nodes.

Let m_t be the number of marks in the subtree of B rooted at v'_t when it is the turn of the surfer at v'_t. Since, at beginning of step i there are at most $2^{2k-2}/k - 2^k + 1$ nodes of B that are marked and $k \geq 4$, $m_0 \leq 2^{2k-2}/k - 2^k + 1 \leq 2^{2k-2}/k$. Note that, for any

$t > 0$, $m_t \leq (m_{t-1} - 1 + \frac{d}{4})/2 \leq (m_{t-1} + \frac{d}{4})/2$. Simply expanding this expression we get that, for any $t > 0$,

$$m_t \leq \frac{m_0}{2^t} + \frac{2^k}{k} \sum_{j=3}^{t+2} 2^{-j} \leq \frac{2^{2k-(t+2)}}{k} + \frac{2^k}{k} \sum_{j=3}^{t+2} 2^{-j}.$$

Therefore, for any $t \geq 2k$:

$$m_t \leq \frac{1}{k} + \frac{2^k}{k} \sum_{j=3}^{t+2} 2^{-j} \leq \frac{2^k + 1}{k}.$$

In particular, at step $i + 2k$ (when it is the turn of the surfer), the surfer is at v'_{2k} which is at distance $k + 1$ from w_0. Hence, $|B^{2k}| \geq 2^{k+1} - 1$ and at most $\frac{2^k+1}{k} < 2^{k+1} - 1$ of its nodes are marked. Hence, neither w_0 nor any node in Q^+ are marked.

From this step, the surfer directly goes to w_k unless it meets an unmarked node, in which case, it goes to it and wins. When the surfer is at w_k and it is its turn, the observer may have marked at most $(2k + 2)\frac{d}{4} \leq \frac{kd}{2} + \frac{d}{2} \leq 2^{k-1} + \frac{2^{k-1}}{k}$ nodes in Q^+. Since $|Q^+| = (d + 1)k = 2^k + k$ and $k \geq 4$, at least one neighbor of w_k is not marked yet and the surfer wins. □

Theorem 48 implies that, for any online strategy $\mathcal{S}$, $\rho(\mathcal{S}) = \Omega(\Delta)$. Recall that the basic strategy $\mathcal{B}$, that marks all unmarked neighbors of the surfer at each step, is an online strategy. $\mathcal{B}$ has trivially competitive ratio $\rho(\mathcal{B}) = O(\Delta)$. Hence, no online winning strategy has better competitive ratio than the basic strategy up to a constant factor. In other words:

Corollary 4. *The best competitive ratio of online winning strategies is* $\Theta(\Delta)$*, where* Δ *is the maximum degree.*

As mentioned in the introduction, any online strategy is connected and therefore, for any graph G and $v_0 \in V(G)$ we have that $\mathrm{csn}(G, v_0) \leq \mathrm{on}(G, v_0)$. Moreover, we recall that, for any tree T and for any $v_0 \in V(T)$, $\mathrm{csn}(T, v_0) = \mathrm{sn}(T, v_0)$ [FGJM+12]. Hence, the previous theorem shows that there might be an arbitrary gap between $\mathrm{csn}(G, v_0)$ and $\mathrm{on}(G, v_0)$.

6.4 Conclusion

In this chapter, we studied the cost of connectivity of the *Surveillance game*. Despite our results, the main question remains open. Can the difference or the ratio between the connected surveillance number of a graph and its surveillance number be bounded by some constant? While we were unable to prove such question, designing examples where this is not true seems very challenging.

On the one hand, if the cost of connectivity is still open for the *connected Surveillance game*, on the other hand, we successfully showed that this gap can be arbitrarily large for the online variant even when compared to the connected variant. The family of trees given in Section 6.3 implicitly shows that, in order to guarantee that the observer wins the game every time, the best strategy is the basic one. While this is unfortunate since the basic strategy might use a lot of marks compared to the best non-online strategy, it is fortunate that the basic strategy can be computed in linear time.

Chapter 7

Fractional Turn-by-Turn Pursuit-Evasion Games

In this chapter, we propose a framework that models several turn-by-turn pursuit-evasion games. This framework, based on linear programming techniques, can be used to define fractional versions of these games. In a fractional version of a pursuit-evasion game, we allow players to play using fractions of tokens. For example, in the fractional version of the *Cops and Robbers* game, both cops and robbers can be split into fractions. That is, a vertex might be occupied by one third of a cop while another vertex might be occupied by two thirds of a cop. We also propose an algorithm, that can be used with any game fitting in this framework, to decide if the pursuer[1] has a winning strategy.

7.1 Description of a Turn-by-Turn Pursuit-Evasion Game

In this section, we present a general game that two players, denoted by $\mathcal{C}$ (the pursuer) and $\mathcal{R}$ (the evader), play on a directed graph $G = (V, E)$ with $n \in \mathbb{N}$ nodes. Let $V = \{1, \ldots, n\}$.

In order to formally describe a fractional turn-by-turn pursuit-evasion game, or simply combinatorial game, we need to introduce some notation. For any vector $x \in \mathbb{R}^n$ and for any $1 \leq i \leq n$, let x_i be the i^{th} coordinate of x, i.e., $x = (x_1, \ldots, x_n)$. The concatenation of two vectors $x, y \in \mathbb{R}^n$ is denoted by $(x, y) = (x_1, \ldots, x_n, y_1, \ldots, y_n) \in \mathbb{R}^{2n}$. The sum of two vectors $x, y \in \mathbb{R}^n$ is denoted by $x+y = (x_1+y_1, \ldots, x_n+y_n) \in \mathbb{R}^n$. More generally, the (Minkowski) sum of two subsets $A, B \subseteq \mathbb{R}^n$ is denoted by $A+B = \{a+b \mid a \in A, b \in B\}$.

The game involves two players, $\mathcal{C}$ and $\mathcal{R}$, that play alternatively on an n-node graph. A *configuration* of the game is represented by a vector $(c, r) \in \mathbb{R}^{2n}_+$ where c and r belong to $\mathbb{R}^n_+$. Intuitively, the i^{th} coordinate of c (resp., of r) represents the amount of tokens of player $\mathcal{C}$ (resp., of player $\mathcal{R}$) on the node $v_i \in V(G)$, $1 \leq i \leq n$. When it is its turn, one player can perform a move, that is, it can modify the current configuration of the game by following some rules described below. Given a configuration $(c, r) \in \mathbb{R}^{2n}_+$, player $\mathcal{C}$ (resp., of player $\mathcal{R}$) can only modify c (resp., r).

Before formally describing the game, we introduce some definitions. The moves of the players will be defined by the following operators.

[1] The pursuer is the observer, cops, devil or guards in the games mentioned in Chapters 5 and 6.

- Let $\mathcal{X}_\mathcal{C} \subseteq \mathbb{R}^n$ and $\mathcal{X}_\mathcal{R} \subseteq \mathbb{R}^n$ be any two convex sets containing 0_n and defined by a polynomial (in n) number of constraints.
- Let Δ_G be a set of stochastic matrices defining by G as follows:

$$\Delta_G = \left\{ [\alpha_{i,j}]_{1\leq i,j\leq n} \;\middle|\; \begin{array}{c} \forall 1 \leq i,j \leq n, \alpha_{i,j} \geq 0, \text{ and} \\ \forall j \leq n, \sum_{1\leq i\leq n} \alpha_{i,j} = 1, \text{ and} \\ \text{if } \{i,j\} \notin E(G) \text{ then } \alpha_{i,j} = 0 \end{array} \right\}$$

  Note that Δ_G is convex and contains the identity matrix.

To understand the intuition behind any matrix in Δ_G, assume that a player has put some tokens on the vertices of G and let $x \in \mathbb{R}^n$ be the vector representing these tokens, i.e., x_i is the amount of tokens on node $v_i \in V(G)$, $1 \leq i \leq n$. Then, for any $\delta \in \Delta_G$, $\delta x \in \mathbb{R}^n$ represents the state after some tokens have have moved (depending on δ) along edges of G. More precisely, for any $1 \leq i,j \leq n$, $\delta_{i,j}$ represents the fraction of tokens initially present in $v_j \in V(G)$ that moved along $\{v_j, v_i\} \in E(G)$ to reach $v_i \in V(G)$.

On the other hand, the vectors in $\mathcal{X}_\mathcal{C}$ and $\mathcal{X}_\mathcal{R}$ will be used to add or remove tokens from nodes of G. For any $y \in \mathcal{X}_\mathcal{C}$ (or $y \in \mathcal{X}_\mathcal{R}$), $x + y$ represents the new state after some tokens have been added or removed to the configuration x. More precisely, for any $1 \leq i,j \leq n$, y_i is the variation of tokens on node v_i (without considering the movement of the tokens along edges incident to v_i).

Now, let us define some particular configurations that will be used to precise a Fractional game:

- let $\mathcal{V} \subseteq \mathbb{R}_+^{2n}$ be a non empty polytope with number of facets polynomial in n. $\mathcal{V}$ is called the set of *valid configurations*;
- let $\mathcal{I} \subseteq \mathcal{V}$ be any non empty set. $\mathcal{I}$ is the set of *initial configurations*;
- let $\mathcal{W}_\mathcal{C} \subseteq \mathcal{C}$ be a polytope with number of facets polynomial in n. This is the set of *winning configurations for* $\mathcal{C}$;
- let $\mathcal{W}_\mathcal{R} = \mathbb{R}_+^{2n} \setminus \mathcal{V}$. This is the set of *winning configurations for* $\mathcal{R}$.
- Let $F \in \mathbb{N}$ be the maximum number of turns the game is allowed to last.
- Finally, let $Last \in \{\mathcal{C}, \mathcal{R}\}$ be the player that wins if the game lasts more than F turns.

Note that if $\mathcal{W}_\mathcal{C} \cup \mathcal{W}_\mathcal{R}$ is empty, then the game will always be won by player *Last*. Now, we are ready to formally define the general game with parameters $\{\mathcal{V}, \mathcal{I}, \mathcal{W}_\mathcal{C}, \mathcal{X}_\mathcal{C}, \mathcal{X}_\mathcal{R}, \Delta_G, F, Last\}$. Note that this is a perfect information game.

1. Initially, $\mathcal{C}$ chooses any vector $c_0 \in \mathbb{R}_+^n$ such that there exists $r \in \mathbb{R}_+^n$ with $(c_0, r) \in \mathcal{I}$. Then, $\mathcal{R}$ chooses any vector $r_0 \in \mathbb{R}_+^n$ such that $(c_0, r_0) \in \mathcal{I}$. $(c_0, r_0) \in \mathcal{I}$ is then the *initial configuration* of the game.
   - If $(c_0, r_0) \in \mathcal{W}_\mathcal{C}$, then player $\mathcal{C}$ wins and the game is over.
   - Else, if $F = 0$, then player *Last* wins and the game is over.

   Otherwise, at each turn $t \geq 1$, there are two steps:

2. First, player $\mathcal{C}$ chooses $\delta \in \Delta_G$ and $x \in \mathcal{X}_\mathcal{C}$ such that $y = (\delta c_{t-1} + x, r_{t-1}) \in \mathcal{V}$. Then, player $\mathcal{C}$ moves to the configuration $(c_t, r_{t-1}) = y$.

   - If $(c_t, r_{t-1}) \in \mathcal{W}_\mathcal{C}$, then player $\mathcal{C}$ wins and the game is over after t turns.

3. Otherwise, $\mathcal{R}$ chooses $\delta \in \Delta_G$ and $x \in \mathcal{X}_\mathcal{R}$ such that $y = (c_t, \delta r_{t-1} + x) \notin \mathcal{W}_\mathcal{C}$. Note that, because $I_{n\times n}$ (identity matrix) is in Δ_G and $0_n \in \mathcal{X}_\mathcal{R}$, then there always exists such y. Then, player $\mathcal{R}$ moves to the configuration $(c_t, r_t) = y$.

   - if $y \notin \mathcal{V}$, i.e., if $y \in \mathcal{W}_\mathcal{R}$, then player $\mathcal{R}$ wins and the game is over after t turns.
   - else, if $t \geq F$, then player *Last* wins and the game is over.
   - Else, the next turn $t + 1$ starts (GOTO 2).

Note that the game is not symmetric in the sense that the role of both players cannot be exchanged. In particular, the set of configurations $\mathcal{W}_\mathcal{C}$ in which $\mathcal{C}$ wants to enter is convex, while the set of configurations $\mathcal{W}_\mathcal{R}$ wants to go is the complementary of the convex set of the valid configurations for $\mathcal{C}$. Moreover, only one player *Last* wins if it can avoid this during a sufficient number F of turns.

A *winning strategy* for player $\mathcal{C}$ consists of a vector c_0 and a function $\sigma : \mathbb{R}^{2n} \to \mathcal{X}_\mathcal{C} \times \Delta_G$ that allows player $\mathcal{C}$ to win whatever be the behavior of player $\mathcal{R}$. That is, player $\mathcal{C}$ chooses c_0 as initial vector, and then, at each turn t, it moves to $(\delta c_{t-1} + x, r_{t-1})$ where $(x, \delta) = \sigma((c_{t-1}, r_{t-1}))$. Following this process, player $\mathcal{C}$ must win in any execution of the game.

In the next section, Section 7.2, we propose an algorithm to decide if the pursuer wins for any game that can be described with this framework. In a semi-fractional game, only the pursuer is allowed to use fractions of tokens, while the evader must use whole integral tokens. We study the gap between semi-fractional and fractional games, in Section 7.3. We show that for some games the resources used by the pursuer are the same in the fractional or in the semi-fractional version. That is, given that the pursuer is playing in a fractional manner, allowing the evader to play in a fractional manner does not help the pursuer. Albeit the results in Section 7.3 are valid for any turn-by-turn pursuit-evasion games mentioned up to this point, they are not valid for every possible game that can be modeled with this framework. Then, in Section 7.4, we focus on some particular games by presenting some results for the *Cops and Robbers* game, the *Angel Problem* and the *Surveillance game*. Finally, in Section 7.5, we conclude this chapter with a discussion about the results found in it.

7.2 Algorithm to Compute a Winning Strategy for player $\mathcal{C}$

In this section, we describe an algorithm that given a game $\{\mathcal{V}, \mathcal{I}, \mathcal{W}_\mathcal{C}, \mathcal{X}_\mathcal{C}, \mathcal{X}_\mathcal{R}, \Delta_G, F,$ *Last*$\}$, decides whether there is a winning strategy for player $\mathcal{C}$.

Roughly, this is done by starting with a set C of configurations which are winning for $\mathcal{C}$ in t turns, meaning that starting the game from any configuration in this set, the game can always be won by $\mathcal{C}$ in at most t turns, and computing a set $C' \supseteq C$. This set C' is such that any configuration in C' is winning for $\mathcal{C}$ in at most $t + 1$ turns. Then, we iterate this process until we get a set C^* such that any configuration in C^* is winning for $\mathcal{C}$ in at most F turns.

To formally state the algorithm, let us define the following sets.

- For any $t \in \mathbb{N}^*$, let $\mathcal{C}_t \subseteq \mathcal{V}$ be the set of configurations such that, for any configuration $m \in \mathcal{C}_t$, there is a strategy with initial configuration m that allows player $\mathcal{C}$ to win in at most t turns. That is, there is a winning strategy for $\mathcal{C}$ in the game $\{\mathcal{V}, \mathcal{C}_t, \mathcal{W}_\mathcal{C}, \mathcal{X}_\mathcal{C}, \mathcal{X}_\mathcal{R}, \Delta_G, t, \mathit{Last}\}$.
- Let $\mathcal{R}_0 = \mathcal{W}_\mathcal{C}$ and, for any $t \in \mathbb{N}^*$, let $\mathcal{R}_t \subseteq \mathcal{V}$ be the set of configurations m such that for every move of player $\mathcal{R}$ from m to m' we have that $m' \in \mathcal{C}_t$. That is, even when the first player to play is $\mathcal{R}$, we have that $\mathcal{C}$ wins if the starting configuration is one in $\mathcal{R}_t$.

Starting from $\mathcal{R}_0 = \mathcal{W}_\mathcal{C}$, our algorithm iteratively, for any $0 < t \leq F$, build $\mathcal{C}_t$ from $\mathcal{R}_{t-1}$ and $\mathcal{R}_t$ from $\mathcal{C}_t$. Then, the desired strategy exists if and only if there is $c_0 \in \mathbb{R}^n$ such that for all $r \in \mathbb{R}^n$ with $(c_0, r) \in \mathcal{I}$ then $(c_0, r) \in \mathcal{C}_F$.

The remaining part of this section is devoted to the formal description of the algorithm and its proof.

Lemma 49. *For all $t \in \mathbb{N}$:*

$$\mathcal{C}_{t+1} = \{(c, r) \in \mathcal{V} \mid \exists x \in \mathcal{X}_\mathcal{C}, \exists \delta \in \Delta_G, (\delta c + x, r) \in \mathcal{R}_t\}.$$

Proof. Let $R = \{(c, r) \in \mathbb{R}^{2n} \mid \exists x \in \mathcal{X}_\mathcal{C}, \delta \in \Delta_G, (\delta c + x, r) \in \mathcal{R}_t\} \cap \mathcal{V}$.

For any $m = (c, r) \in R \subseteq \mathcal{V}$, we show that there is a strategy for $\mathcal{C}$ to win the game in at most $t + 1$ turns starting from m. Indeed, by definition of R, there are $x \in \mathcal{X}_\mathcal{C}$ and $\delta \in \Delta_G$ such that $c' = \delta c + x$ with $(c', r) \in \mathcal{R}_t \subseteq \mathcal{V}$. Then, in configuration (c, r), player $\mathcal{C}$ moves to (c', r). Since $(c', r) \in \mathcal{R}_t$, for any move of player $\mathcal{R}$, say it moves to (c', r'), then $(c', r') \in \mathcal{C}_t$ by definition of $\mathcal{R}_t$. Finally, by definition of $\mathcal{C}_t$, there is a strategy that allows $\mathcal{C}$ to win in at most t turns starting from (c', r'). Hence, $R \subseteq \mathcal{C}_{t+1}$.

Reciprocally, let $(c, r) \in \mathcal{C}_{t+1} \subseteq \mathcal{V}$. By definition, there is a strategy σ that allows $\mathcal{C}$ to win in at most $t + 1$ turns starting from (c, r). Let $(x, \delta) = \sigma((c, r)) \in \mathcal{X}_\mathcal{C} \times \Delta_G$ and $c' = \delta c + x$. Since σ is winning, whatever be the move (c', r') of player $\mathcal{R}$ from (c', r), player $\mathcal{C}$ wins in at most t turns starting from (c', r'). Hence, $(c', r) \in \mathcal{R}_t$. Therefore, $(c, r) \in R$ and $\mathcal{C}_{t+1} \subseteq R$. □

In Lemma 50, we describe how a system of linear inequalities describing $\mathcal{C}_{t+1}$ can be obtained through a system of linear inequalities describing $\mathcal{R}_t$. For that we first construct an intermediate system of linear inequalities R that is equivalent to $\mathcal{C}_{t+1}$ but which has several auxiliary variables and, then, we proceed to eliminate those variables obtaining the final set $R' = \mathcal{C}_{t+1}$. This process, however, is very costly blowing the complexity of the algorithm. If, on the one hand, this elimination process seems rather unnecessary since it does help describing $\mathcal{C}_{t+1}$, on the other hand, with the auxiliary variables we are unable to correctly construct $\mathcal{R}_{t+1}$ from $\mathcal{C}_{t+1}$ since one of the hypothesis of Lemma 52 is that $\mathcal{R}_{t+1} \subseteq \mathbb{R}^{2n}_+$.

Lemma 50. *Let $t \geq 0$ and assume that $\mathcal{R}_t \subseteq \mathbb{R}^{2n}_+$ is a convex set described by ℓ linear inequalities and $2n$ variables. Then, there is an algorithm that computes a set of*

$$O\left(4\left(\frac{\ell + O(\max\{p(n), n^2\})}{4}\right)^{2^{2n+n^2}}\right)$$

linear inequalities and $2n$ variables describing $\mathcal{C}_{t+1}$, where $p(n)$ is the maximum between the sizes of the system of linear inequalities describing $\mathcal{V}$ and $\mathcal{X}_\mathcal{C}$.

Proof. Let us consider the following convex set R.

$$
\begin{array}{lll}
(c', r) = (c'_1, \ldots, c'_n, r_1, \ldots, r_n) \in \mathbb{R}^{2n}_+ & & \\
\text{Subject to} & & \\
(c', r) \in \mathcal{V} & & (1) \\
c_i = x_i + a_{i,i} + \sum_{1 \le j \le n, \{i,j\} \in E(G)} a_{i,j} & \forall 1 \le i \le n & (2) \\
(c_1, \ldots, c_n, r_1, \ldots, r_n) \in \mathcal{R}_t & & (3) \\
(x_1, \ldots, x_n) \in \mathcal{X}_{\mathcal{C}} & & (4) \\
c'_i = a_{i,i} + \sum_{1 \le j \le n, \{i,j\} \in E(G)} a_{j,i} & \forall 1 \le i \le n & (5.a) \\
a_{i,j} \ge 0 & \forall 1 \le i, j \le n & (5.c) \\
a_{j,i} = 0 & \forall i \ne j \in [1,n]^2, \{j,i\} \notin E(G) & (5.b)
\end{array}
$$

$\mathcal{V}$ and $\mathcal{X}_{\mathcal{C}}$ are convex sets defined by polynomial (in n) number of linear inequalities. Therefore, $p(n) = O(n^k)$ for some fixed k.

By hypothesis, $\mathcal{R}_t$ is a convex set defined by ℓ linear inequalities. Since there are at most $O(n^2)$ linear equations (2) and (5) with $O(n^2)$ new variables, the above linear program has a total of $\ell + O(\max\{p(n), n^2\})$ linear inequalities and $O(n^2)$ variables.

Moreover, given the set of inequalities defining $\mathcal{V}$, $\mathcal{X}_{\mathcal{C}}$ and $\mathcal{R}_t$, the above set of inequalities can be computed in time $O(\ell + \max\{p(n), n^2\})$. Note that if ℓ can be bounded by a polynomial in n and t then R can be constructed in polynomial-time (in n and t).

Now, let us show that $\mathcal{C}_{t+1}$ can be described by the above system of linear inequalities by projecting R over the variables $c'_1, \ldots, c_n$ and $r_1, \ldots, r_n$. That is, (c', r) belongs to R projected into $c'_1, \ldots, c_n$ and $r_1, \ldots, r_n$ if and only if $(c', r) \in \mathcal{C}_{t+1}$. Indeed,

(c', r) belongs to R projected into $c'_1, \ldots, c_n$ and $r_1, \ldots, r_n$ if and only if there exist values of c_i, x_i and $a_{i,j}$, for $1 \le i, j \le n$, such that $(c', c, x, a, r) \in R$.

$\Leftrightarrow$ $(c', r) \in \mathcal{V}$ and there exist $x = (x_1, \ldots, x_n) \in \mathcal{X}_{\mathcal{C}}$ and $A = [a_{i,j}]_{1 \le i,j \le n} \in \mathbb{R}^+_{n \times n}$ such that $(A1_n + x, r) \in \mathcal{R}_t$, where $1_n = (1, \ldots, 1) \in \mathbb{R}^n$, and for all $i \le n$, $a_{i,i} + \sum_{1 \le j \le n, \{i,j\} \in E(G)} a_{j,i} = c_i$ and $a_{j,i} = 0$ for any $j \ne i$, $\{j,i\} \notin E(G)$.

$\Leftrightarrow$ $(c', r) \in \mathcal{V}$ and there exist $x \in \mathcal{X}_{\mathcal{C}}$ and $\delta = [\alpha_{i,j}]_{1 \le i,j \le n} = [\frac{a_{i,j}}{c_j}]_{1 \le i,j \le n} \in \Delta_G$ such that $(\delta c' + x, r) \in \mathcal{R}_t$.

$\Leftrightarrow$ $(c', r) \in \mathcal{C}_{t+1}$, by Lemma 49.

The set R, however, has several variables that are auxiliary. It is necessary to eliminate the variables c_i, x_i and $a_{i,j}$ for all $1 \le i, j \le n$. For this we successively use the the Fourier–Motzkin elimination method [Sch98] on these variables. See Remark 5 for a brief discussion on how this method works. Since there are $2n + n^2$ variables in total that we want to eliminate, R' obtained after eliminating all c_i, x_i and $a_{i,j}$ variables might have a number of linear inequalities equal to

$$
O\left(4\left(\frac{\ell + O(\max\{p(n), n^2\})}{4}\right)^{2^{2n+n^2}}\right).
$$

However, since there are only $2n$ variables in R' and that R' is a projection of R into $c'_1, \ldots, c'_n$ and $r_1, \ldots, r_n$, we have that $R' = \mathcal{C}_{t+1}$. □

Remark 5. For the sake of completeness we briefly illustrate the Fourier–Motzkin elimination method. Let $A_{\ell \times n} x \le b$ be a system of linear inequalities. Assume that we want

to eliminate the last variable of the vector x from this system. Let x_n be this variable. We first rewrite all inequalities such that $a_{i,1} \neq 0$ in order to isolate x_n. That is, every inequality such that the coefficient x_n is not 0 is rewritten as (Type 1) $x_n \geq$ "something" or (Type 2) $x_n \leq$ "something". Note that the coefficient of x_n is 1. Then, there are two cases to consider:

- There are only inequalities of the form $x_n \geq$ "something" or there are only inequalities of the form $x_n \leq$ "something". In this case we simply remove all these inequalities from $Ax \leq b$.
- If there are both types of inequalities, then we combine each inequality of (Type 1) with each inequality of (Type 2). That is, for each pair of inequalities $x_n \leq$ "somethingA" and $x_n \geq$ "somethingB", we add the inequality "somethingB" $\leq$ "somethingA" to $Ax \leq b$. Then, we remove all inequalities such that the coefficient of x_n is non-zero.

This method guarantees that, after eliminating a variable, the result is a system of inequalities $A'_{\ell' \times n-1} x' \leq b'$ such that $A'x' \leq b'$ has a solution if and only if $Ax \leq b$ has a solution. Moreover, if x' is a solution for $A'_{\ell' \times n-1} x' \leq b'$ then there is $x = (x_1, \ldots, x_n)$ with $(x_1, \ldots, x_{n-1}) = (x'_1, \ldots, x'_{n-1})$ such that x is a solution to $Ax \leq b$. In other words, we this process projects the set described by $Ax \leq b$ into its first $n-1$ variables.

While we remove some inequalities, a single execution of this method, however, might add $\ell^2/4$ new inequalities, where ℓ is the number of initial inequalities. Hence, in order to eliminate d variables, we might add $4(\ell/4)^{2^d}$ inequalities.

The proof of next lemma is similar to the proof of Lemma 49, by exchanging the role of $\forall$ and $\exists$ and the role of both players.

Lemma 51. $\mathcal{R}_t = \{(c, r) \in \mathbb{R}^{2n} \mid \forall x \in \mathcal{X}_\mathcal{R}, \forall \delta \in \Delta_G, (c, \delta r + x) \in \mathcal{C}_t\} \cap \mathcal{V}$.

Let $x \cdot y$ denote the scalar product of x and y, and let x^T denote the transpose of x.

Lemma 52. *Let $t \geq 0$ and assume that $\mathcal{C}_t \subseteq \mathbb{R}^{2n}_+$ is a convex set described by ℓ linear inequalities and $2n$ variables. Then, there is a polynomial-time algorithm in ℓ and n that computes a set of at most ℓ linear inequalities and $2n$ variables describing $\mathcal{R}_t$.*

Proof. By the hypothesis, there exist $A \in \mathbb{R}^{\ell \times 2n}$ and $b = (b_1, \ldots, b_\ell) \in \mathbb{R}^\ell$ such that $\mathcal{C}_t = \{m \in \mathbb{R}^{2n}_+ \mid Am \leq b\}$.

For any $1 \leq i \leq \ell$, let $(z_{i,1}, \ldots, z_{i,n}, a_{i,1}, \ldots, a_{i,n})$ be the i^{th} row of A. Let $A_i = (a_{i,1}, \ldots, a_{i,n})$.

- Let $b'_i = \max_{x \in \mathcal{X}_\mathcal{R}} \{A_i x\}$ and let $X_i \in \operatorname{argmax}_{x \in \mathcal{X}_\mathcal{R}} \{A_i x\}$. This is computable in polynomial-time in n since $\mathcal{X}_\mathcal{R}$ is a convex set defined by a polynomial number of constraints.
- For any $u \in V(G)$, let $u_i \in \operatorname{argmax}_{v \in N(u)} \{a_{i,v}\}$. Let $\delta_i = [\alpha_{v,u}]_{1 \leq u,v, \leq n}$ such that, for any $1 \leq v, u \leq n$, $\alpha_{v,u} = 1$ if $v = u_i$ and $\alpha_{v,u} = 0$ otherwise. Clearly, $\delta_i \in \Delta_G$.

Let us consider the following convex set R.

$$
\begin{aligned}
& (c, r) = (c_1, \ldots, c_n, r_1, \ldots, r_n) \in \mathbb{R}^{2n}_+ \\
\text{Subject to} \quad & \\
& (c, r) \in \mathcal{V} \\
& (z_{i,1}, \ldots, z_{i,n}, A_i \delta_i) \cdot (c, r) \leq b_i - b'_i \qquad , \forall 1 \leq i \leq \ell
\end{aligned}
$$

Since $\mathcal{V}$ is a convex set defined by a size polynomial in n and $\mathcal{C}_t$ is a convex set described ℓ linear inequalities, the above linear system has size polynomial in ℓ and n and can be computed in polynomial-time (in ℓ and n).

It remains to show that:

Claim 16. $R = \mathcal{R}_t$.

Let $(c, r) \in \mathcal{R}_t$. By Lemma 51, $(c, r) \in \mathcal{V}$ and, for any $\delta \in \Delta_G$ and $x \in \mathcal{X}_\mathcal{R}$, $(c, \delta r + x) \in \mathcal{C}_t$. Then, for any $1 \leq i \leq \ell$, $(c, \delta_i r + X_i) \in \mathcal{C}_t$. In other words, $A(c, \delta_i r + X_i)^T \leq b$. In particular, $(z_{i,1}, \ldots, z_{i,n}, A_i) \cdot (c, \delta_i r + X_i) = (z_{i,1}, \ldots, z_{i,n}, A_i\delta_i) \cdot (c, r) + A_i X_i = (z_{i,1}, \ldots, z_{i,n}, A_i\delta_i) \cdot (c, r) + b'_i \leq b_i$. Hence, $(c, r) \in R$.

Let $(c, r) \in R$. Then, $(c, r) \in \mathcal{V}$. Let $\delta = [\alpha'_{i,j}]_{1 \leq i,j \leq n} \in \Delta_G$ and $x \in \mathcal{X}_\mathcal{R}$. We show that $(c, \delta r + x) \in \mathcal{C}_t$. More precisely, we show that $A \cdot (c, \delta r + x) \leq b$. Let $1 \leq i \leq \ell$. Then,

$$(z_{i,1}, \ldots, z_{i,n}, A_i) \cdot (c, \delta r + x) = (z_{i,1}, \ldots, z_{i,n})c + A_i \delta r + A_i x.$$

Since $X_i \in \operatorname{argmax}_{x \in \mathcal{X}_\mathcal{R}} \{A_i x\}$, we have $b'_i = A_i X_i \geq A_i x$. Hence,

$$(z_{i,1}, \ldots, z_{i,n}, A_i) \cdot (c, \delta r + x) \leq (z_{i,1}, \ldots, z_{i,n})c + A_i \delta r + b'_i.$$

Moreover, because $(c, r) \in R$, for any $1 \leq i \leq \ell$, $(z_{i,1}, \ldots, z_{i,n}, A_i\delta_i) \cdot (c, r) \leq b_i - b'_i$. Hence,

$$(z_{i,1}, \ldots, z_{i,n})c + A_i \delta_i r \leq b_i - b'_i.$$

To show that $(z_{i,1}, \ldots, z_{i,n}, A_i) \cdot (c, \delta r + x) \leq b_i$, it remains to prove that $A_i \delta r \leq A_i \delta_i r$

On the one hand,

$$A_i \delta r = \sum_{1 \leq j \leq n} a_{i,j} \sum_{1 \leq k \leq n} \alpha'_{j,k} r_k = \sum_{1 \leq k \leq n} r_k \sum_{1 \leq j \leq n} a_{i,j} \alpha'_{j,k}.$$

Since, for any $1 \leq k \leq n$, $\sum_{1 \leq j \leq n} \alpha'_{j,k} = 1$ and for all $1 \leq j, k \leq n$, $\alpha'_{j,k} \geq 0$ and $r_k \geq 0$, we get that $A_i \delta r \leq \sum_{1 \leq k \leq n} r_k \max_{1 \leq j \leq n} a_{i,j}$.

On the other hand,

$$A_i \delta_i r = \sum_{1 \leq k \leq n} r_k \sum_{1 \leq j \leq n} a_{i,j} \delta_{j,k}.$$

Recall that, by definition, there is exactly one $1 \leq j \leq n$ such that $\delta_{j,k} = 1$, and such that $a_{i,j} = \max_{1 \leq j' \leq n} a_{i,j'}$, and $\delta_{j,k} = 0$ for all the $(n-1)$ other values of j. Therefore, $A_i \delta_i r = \sum_{1 \leq k \leq n} r_k \max_{1 \leq j \leq n} a_{i,j}$. Thus, we got the result, i.e., for any $1 \leq i \leq \ell$, $(z_{i,1}, \ldots, z_{i,n}, A_i) \cdot (c, \delta r + x) \leq b_i$.

Therefore, $A(c, \delta r + x)^T \leq b$ and, by Lemma 51, $(c, r) \in \mathcal{R}_t$. Hence, $R = \mathcal{R}_t$. □

Hence, by applying Lemma 50 and Lemma 52 successively, we are able to construct $\mathcal{C}_F$ from $\mathcal{R}_0 = \mathcal{W}_\mathcal{C}$. However, this construction might take more than polynomial time.

7.3 Semi-Fractional and Integral Games

In this section, we define the semi-fractional and integral games related to the general fractional game studied above. We then show that fractional games and semi-fractional games are equivalent under a weak hypothesis. In particular, this implies that when the game is related to an optimization problem, the fractional game provides a bound for the integral game.

Let $\mathcal{G} = \{\mathcal{V}, \mathcal{I}, \mathcal{W}_\mathcal{C}, \mathcal{X}_\mathcal{C}, \mathcal{X}_\mathcal{R}, \Delta_G, F, Last\}$ be a fractional game as defined in Section 7.1. We defined the corresponding *integral game* as

$$\mathcal{G}_{int} = \{\mathcal{V}, \mathcal{I} \cap \mathbb{N}^{2n}, \mathcal{W}_\mathcal{C} \cap \mathbb{N}^{2n}, \mathcal{X}_\mathcal{C} \cap \mathbb{N}^n, \mathcal{X}_\mathcal{R} \cap \mathbb{N}^n, \Delta_G \cap \mathbb{N}_{n\times n}, F, Last\},$$

where the rules of the game are exactly the same as before. That is, we restrict all configurations and moves to be integral. In particular, this means that the sets of configurations are not convex any more (actually, the set of valid configuration $\mathcal{C}$ is still convex but only configurations in $\mathcal{C} \cap \mathbb{N}^{2n}$ can be achieved) and that the algorithm of previous section does not apply any more.

Let us also denote the corresponding *semi-fractional game* by

$$\mathcal{G}_{sf} = \{\mathcal{V}, \mathcal{I} \cap (\mathbb{R}^n \times \mathbb{N}^n), \mathcal{W}_\mathcal{C}, \mathcal{X}_\mathcal{C}, \mathcal{X}_\mathcal{R} \cap \mathbb{N}^n, (\Delta_G^\mathcal{C} = \Delta_G, \Delta_G^\mathcal{R} = \Delta_G \cap \mathbb{N}_{n\times n}), F, Last\}.$$

Note that we distinguished the two sets $\Delta_G^\mathcal{C}$ and $\Delta_G^\mathcal{R}$. Indeed, the game proceeds as before, but player $\mathcal{R}$ is constrained to move only on integral configurations. That is, a move for player $\mathcal{R}$ is to choose $\delta \in \Delta_G^\mathcal{R} = \Delta_G \cap \mathbb{N}_{n\times n}$ and $x \in \mathcal{X}_\mathcal{R} \cap \mathbb{N}^n$ such that $m = (c, \delta r + x) \notin \mathcal{W}_\mathcal{C}$ and then to move from (c, r) to m. On the other hand, player $\mathcal{C}$ is not constrained and its moves are still defined by $\delta \in \Delta_G^\mathcal{C} = \Delta_G$ and $x \in \mathcal{X}_\mathcal{C}$.

The next Lemma directly follows from the definition of the games.

Lemma 53. *Let $\mathcal{G}$ be a fractional game.*

- *Player $\mathcal{C}$ has a winning strategy in $\mathcal{G}$ only if it has a winning strategy in $\mathcal{G}_{sf}$.*
- *Player $\mathcal{C}$ has a winning strategy in $\mathcal{G}_{int}$ only if it has a winning strategy in $\mathcal{G}_{sf}$.*

Proof. Indeed, any winning strategy in $\mathcal{G}$ (resp., in $\mathcal{G}_{int}$) is a winning strategy in $\mathcal{G}_{sf}$. Indeed, the possible moves and initial configurations of $\mathcal{C}$ in $\mathcal{G}$ are still possible in $\mathcal{G}_{sf}$ while the moves (and initial configurations) of $\mathcal{R}$ are more constrained in $\mathcal{G}_{sf}$. On the other hand, the possible moves and initial configurations of $\mathcal{C}$ in $\mathcal{G}_{int}$ are still possible in $\mathcal{G}_{sf}$ while the moves and initial configurations of $\mathcal{R}$ remains the same in $\mathcal{G}_{int}$ and $\mathcal{G}_{sf}$. □

We prove that under a small extra assumption, fractional and semi-fractional games are equivalent. Intuitively, assume that $\mathcal{C}$ can win against any integral strategy of $\mathcal{X}_\mathcal{R}$. Now assume that $\mathcal{X}_\mathcal{R}$ can split each of its tokens into two half-tokens following two distinct strategies. Then, $\mathcal{C}$ will also use half-tokens to win against the strategy of the first half-tokens of $\mathcal{X}_\mathcal{R}$, and the second half of the tokens of $\mathcal{C}$ will win against the strategy followed by the remaining half-tokens of $\mathcal{X}_\mathcal{R}$. By convexity of the moves of $\mathcal{C}$, this is a valid strategy. We propose here another proof, based on the algorithm defined in Section 7.2.

Theorem 54. *If all the vertices of the polytopes $\mathcal{X}_\mathcal{R}$, $\mathcal{X}_\mathcal{C}$ and $\mathcal{W}_\mathcal{C}$ have integral coordinates and if $\mathcal{I} \subseteq \mathbb{R}^n \times \mathbb{N}^n$, then:*

Player $\mathcal{C}$ has a winning strategy in $\mathcal{G}$ if and only if it has a winning strategy in $\mathcal{G}_{sf}$.

Proof. By previous lemma, it is sufficient to prove that if $\mathcal{C}$ has a winning strategy in $\mathcal{G}_{sf}$ then it has a winning strategy in $\mathcal{G}$.

For any $1 \leq t \leq F$, $\mathcal{C}_t$ is defined as in Section 7.2 as the set of configurations from which $\mathcal{C}$ wins in at most t turns in the fractional game. Let $\mathcal{C}_t^{sf} \subseteq \mathcal{C}_t \cap (\mathbb{R}^n \times \mathbb{N}^n)$ be the set of configurations from which $\mathcal{C}$ wins in at most t turns in the semi-fractional game.

Let $\mathcal{R}_0^{sf} = \mathcal{W}_\mathcal{C} \cap (\mathbb{R}^n \times \mathbb{N}^n) = \mathcal{R}_0 \cap (\mathbb{R}^n \times \mathbb{N}^n)$ and, for any any $1 \leq t \leq F$, let $\mathcal{R}_t$ is defined as in Section 7.2 as the set of configurations from which player $\mathcal{R}$ can only enter

in $\mathcal{C}_t$ in the fractional game, i.e., in one (fractional) move. Let $\mathcal{R}_t^{sf} \subseteq \mathcal{V} \cap (\mathbb{R}^n \times \mathbb{N}^n)$ be the set of configurations from which player $\mathcal{R}$ can only enter in $\mathcal{C}_t^{sf}$ in the semi-fractional game, i.e., in one integral move.

Given $X \subseteq \mathbb{R}^{2n}$, let $\mathrm{CH}(X)$ be the convex hull of X.

We prove by induction on t that, for any $1 \leq t \leq F$, $\mathcal{C}_t = \mathrm{CH}(\mathcal{C}_t^{sf})$ and $\mathcal{R}_t = \mathrm{CH}(\mathcal{R}_t^{sf})$.

Let $t \geq 0$, and assume for purpose of induction that $\mathcal{R}_t = \mathrm{CH}(\mathcal{R}_t^{sf})$. This is true for $t = 0$ by definition and because the vertices of $\mathcal{W}_\mathcal{C}$ have integral coordinates. By a proof as the one of Lemma 49, $\mathcal{C}_{t+1}^{sf} = \{(c, r) \in \mathcal{V} \cap (\mathbb{R}^n \times \mathbb{N}^n) \mid \exists x \in \mathcal{X}_\mathcal{C}, \exists \delta \in \Delta_G, (\delta c + x, r) \in \mathcal{R}_t^{sf}\}$. Therefore, because $\mathcal{R}_t = \mathrm{CH}(\mathcal{R}_t^{sf})$ by induction, $\mathcal{C}_{t+1}^{sf} = \{(c, r) \in \mathcal{V} \cap (\mathbb{R}^n \times \mathbb{N}^n) \mid \exists x \in \mathcal{X}_\mathcal{C}, \exists \delta \in \Delta_G, (\delta c + x, r) \in \mathcal{R}_t\}$. And thus, $\mathrm{CH}(\mathcal{C}_{t+1}^{sf}) = \mathcal{C}_{t+1}$ by Lemma 49 and because the vertices of $\mathcal{X}_\mathcal{C}$ and Δ_G have integral coordinates.

Let $t > 0$, and assume for purpose of induction that $\mathcal{C}_t = \mathrm{CH}(\mathcal{C}_t^{sf})$. This is true for $t = 1$ by above paragraph. By the same proof as the one of Lemmas 49 and 51, $\mathcal{R}_t^{sf} = \{(c, r) \in \mathcal{V} \cap (\mathbb{R}^n \times \mathbb{N}^n) \mid \forall x \in \mathcal{X}_\mathcal{R} \cap \mathbb{N}^n, \forall \delta \in \Delta_G^R = \Delta_G \cap \mathbb{N}_{n\times n}, (c, \delta r + x) \in \mathcal{C}_t^{sf}\}$. Therefore, by induction, $\mathcal{R}_t^{sf} = \{(c, r) \in \mathcal{V} \cap (\mathbb{R}^n \times \mathbb{N}^n) \mid \forall x \in \mathcal{X}_\mathcal{R} \cap \mathbb{N}^n, \forall \delta \in \Delta_G^R = \Delta_G \cap \mathbb{N}_{n\times n}, (c, \delta r + x) \in \mathcal{C}_t\}$.

The proof is then the similar as the one of Lemma 52. Recall that $\mathcal{C}_t = \{x \in \mathbb{R}_+^{2n} \mid Am \leq b\}$, and $A_i = (a_{i,1}, \ldots, a_{i,n})$ where $(z_{i,1}, \ldots, z_{i,n}, a_{i,1}, \ldots, a_{i,n})$ be the i^{th} row of A, for any $1 \leq i \leq \ell$.

Note first that, because the vertices of $\mathcal{X}_\mathcal{R}$ have integral coordinates, $b'_i = \max_{x\in\mathcal{X}_\mathcal{R}}\{x \cdot A_i\} = \max_{x\in\mathcal{X}_\mathcal{R}\cap\mathbb{N}^n}\{x \cdot A_i\}$ and therefore, there is $X_i \in \mathbb{N}^n$ such that $X_i \in \mathrm{argmax}_{x\in\mathcal{X}_\mathcal{R}}\{x \cdot A_i\}$. Let also $\delta_i \in \Delta_G \cap \mathbb{N}_{n\times n}$ as defined in the proof of Lemma 52.

By the same proof as the one of Lemma 52, it can be proved that $\mathcal{R}_t^{sf}$ is defined by

$$
\begin{aligned}
& (c, r) = (c_1, \ldots, c_n, r_1, \ldots, r_n) \in \mathbb{R}^n \times \mathbb{N}^n \\
\text{Subject to} \quad & \\
& (c, r) \in \mathcal{V} \\
& (z_{i,1}, \ldots, z_{i,n}, A_i\delta_i) \cdot (c, r) \leq b_i - b'_i \qquad , \forall 1 \leq i \leq \ell
\end{aligned}
$$

Therefore, because $X_i \in \mathbb{N}^n$ (for any $1 \leq i \leq \ell$) and $\delta_i \in \Delta_G \cap \mathbb{N}_{n\times n}$, we get that $\mathcal{R}_t = \mathrm{CH}(\mathcal{R}_t^{sf})$.

Hence, $\mathcal{C}_t = \mathrm{CH}(\mathcal{C}_t^{sf})$. This is easy to conclude because $\mathcal{I} \subseteq \mathbb{R}^n \times \mathbb{N}^n$. □

7.4 Applications in Combinatorial Games

In this section, we discuss how to model some of the turn-by-turn pursuit-evasion games mentioned in this thesis with the framework given in Section 7.1. Moreover, based on the fractional game we show how to construct strategies for the *Angel Problem* and the *Surveillance game* that are winning with high probability and are at most a factor of $\log n$ from the fractional strategy, where n is the order of the graph. On the other hand, for the *fractional Cops and Robbers* game we show that $1 + \epsilon$, $\epsilon > 0$, cops are enough to capture the robber.

Cops and Robbers

The classical *Cops and Robbers* game fits our framework. Indeed, consider the *Cops and Robbers* game played with $k > 0$ cops on a graph $G = (V, E)$ of order n. This game can be defined using the following sets: $\mathcal{I} = \mathcal{V} = \{(c, r) \in \mathbb{R}_+^{2n} \mid \sum r_i = 1, \sum c_i = k\}$, $\mathcal{X}_\mathcal{R} = \mathcal{X}_\mathcal{C} = \{(0, \ldots, 0)\}$, $\Delta_\mathcal{C} = \Delta_\mathcal{R} = \Delta_G$, $\mathit{Last} = \mathcal{R}$ and $\mathcal{W}_\mathcal{C} = \{(c, r) \in \mathcal{V} \mid \forall i \in V, c_i \geq r_i\}$. While

we can limit F to be at most n^{k+1}, since there are at most n^{k+1} possible configurations for the integral game, we leave F undefined. That is, $F = \infty$.

For an easier presentation, let us give an alternative definition of the *semi-fractional Cops and Robbers* game. We consider the following game played with k cops. The cops are tokens that can split themselves and move them along edges. The robber is one un-splittable token that can move along edges.

First, the cops choose a position for themselves, i.e., they choose a vector $\mathcal{C}_0 = (c_1, \ldots, c_n) \in \mathbb{R}^n$ such that $\sum_{i \leq n} c_i = k$, where c_i is the amount of cops at vertex $i \in V$. Let $\mathcal{I} = \{(c, r) \in \mathbb{R}_+^{n+1} \mid r \in V, \sum c_i = k\}$. Then, the robber chooses a vertex $\mathcal{R}_0 \in V$ such that $(\mathcal{C}_0, \mathcal{R}_0) \in \mathcal{I}$. $(\mathcal{C}_0, \mathcal{R}_0)$ is called the starting configuration of the game.

The game is played turn-by-turn. After turn $t \geq 0$, let $(\mathcal{C}_t, \mathcal{R}_t)$ be the current configuration of the game. Then, for any $i \leq n$, the cops can move any portion of c_i on any node $j \in N(i)$, for any $i \leq n$. Formally, the cops choose a left stochastic matrix $A = [\alpha_{i,j}]_{1 \leq i,j \leq n} \in [0,1]_{n \times n}$ (i.e., for any $j \leq n$, $\sum_{i \leq n} \alpha_{i,j} = 1$) such that, for any $1 \leq i, j \leq n$, if $\{i, j\} \notin E$, then $\alpha_{i,j} = 0$. The new "cop-configuration" is $\mathcal{C}_{t+1} = A \cdot \mathcal{C}_t$ (intuitively, a proportion of $\alpha_{i,j}$ cops are moved from v_j to v_i). Then, the robber may move itself to an adjacent node, i.e., $\mathcal{R}_{t+1} \in N[\mathcal{R}_t]$.

The cops win the game if they eventually achieve (after their turn) a configuration $(c_1, \ldots, c_n, j)$ such that $c_j \geq 1$. The robber wins otherwise.

Let $\mathrm{fcn}(G)$ be the smallest k such that the cops have a finite winning strategy, i.e., they can win in a finite number of steps whatever the robber does.

Let $\mathrm{fcn}_\infty(G)$ be the smallest k such that the cops have an infinite winning strategy, i.e., there is a strategy for the cops, such that for any $\epsilon > 0$ and whatever be the strategy of the robber, the cops can achieve (after their turn) a configuration $(c_1, \ldots, c_n, j)$ such that $c_j \geq 1 - \epsilon$.

The following results were obtained by Mazauric, Lamprou and Nisse in personal communication.

Claim 17 (Mazauric *et al.*)**.** *For any graph G, $1 \leq \mathrm{fcn}_\infty(G) \leq \mathrm{fcn}(G) \leq$ fractional dominating number(G)*

Proof. Clearly, from their definition, $1 \leq \mathrm{fcn}_\infty(G) \leq \mathrm{fcn}(G)$. To see that $\mathrm{fcn}(G) \leq$ fractional dominating number(G), let S be a fractional dominating set of G. Assume that $V(G) = \{1, \ldots, n\}$. Then, let s_i be the amount of vertex v_i that is on S. Hence, for all $v \in V$, $\sum_{i \in N[v]} s_i = 1$. Therefore, by placing s_i cops on each vertex i during its positioning, we have that the robber can be captured by the cops in their next move. □

Lemma 55 (Mazauric *et al.*)**.** *There are graphs G such that* $\mathrm{fcn}(G) > 1$

Proof. Consider any graph containing a cordless cycle with four nodes. Consider any fractional strategy with one cop. The robber chooses first a node v such that $N[v]$ contains less than 1 cop. Then, there is a node, not in $N[v]$, where there is at least $\epsilon > 0$ cops. During the next step and remaining on the cycle, the robber can maintain a distance at least one between itself and a proportion $\epsilon' > 0$ cops of these ϵ cops. □

Theorem 56 (Mazauric *et al.*)**.** *For any graph G, $\mathrm{fcn}_\infty(G) = 1$, and for any $\beta > 0$, $\mathrm{fcn}(G) \leq 1 + \beta$. Moreover, there is a finite winning strategy that allows the cops to capture the robber in a linear number of turns.*

Proof. If $G = K_n$, the result is trivial so let us assume that G has minimum degree $\delta < n - 1$. Let us define the following strategy. First, the $k = x_0 = 1 + \beta$, in the case of

$\mathrm{fcn}(G)$, or $k = x_0 = 1$, in the case of $\mathrm{fcn}_\infty(G)$, cops places themselves uniformly at each node (i.e., x_0/n at each node). Then, the robber places itself at some node v. Then, $\delta x_0/n$ cops at the neighbors of v goes to v. At this step, there are $y_1 = (1+\delta)x_0/n$ cops at the same node v as the robber. The remaining amount of the cops is $x_1 = x_0 - y_1 = (1-\frac{1+\delta}{n})k$.

By induction on $t \geq 0$, assume that $y_t = k - x_t$ cops occupy the same node v as the robber and it is the turn of the robber. Moreover, the remaining amount of cops is $x_t = (1 - \frac{1+\delta}{n})^t k$. Now, the robber moves to a node w adjacent to v. Then, the y_t cops on v move to w and there are two cases to consider:

1. if the x_t remaining cops are not uniformly placed (i.e., x_t/n at each node), they move to achieve such a position. This can be done, in one step, by moving the cops along a spanning tree of G rooted in w, where each vertex moves to its parent an amount of cops that is proportional to the number of its descendants in the spanning tree.

2. else, $\delta x_t/n$ cops at the neighbors of w goes to w. Moreover, before this move, except the y_t cops there are also x_t/n cops at w. Therefore, after this step, there are $y_{t+1} = (1+\delta)x_t/n + y_t$ cops at the same node w as the robber.

   Hence,
   $$x_{t+1} = x_t - \frac{(1+\delta)\,x_t}{n} = x_t\left(1 - \frac{1+\delta}{n}\right) = k\left(1 - \frac{1+\delta}{n}\right)^{t+1}$$
   and
   $$y_{t+1} = k - x_{t+1}.$$

The result follows, essentially, from the fact that $\lim_{t\to\infty} y_t = 1$ when $\beta = 0$ and that there exists $t > 0$ such that $y_t \geq 1 + \beta$ when $\beta > 0$. □

Surveillance Game

The classical *Surveillance game* also fits our framework. Consider an observer that can mark at most k vertices at each turn and assume that the game is played on a graph $G = (V, E)$ with $V = \{1, \ldots, n\}$ where the initial vertex is vertex 1. Then, the *fractional Surveillance game* can be defined with the help of the following sets:

$$\begin{aligned}
\mathcal{I} &= \{(c_1, \ldots, c_n, r_1, \ldots, r_n) \mid c_1 = 1, r_1 = 1, \forall i \in V(G) \setminus \{1\}, c_i = 0, r_i = 0\}, \\
\mathcal{V} &= \left\{(c, r) \in \mathbb{R}_+^{2n} \mid \forall i \in V(G), c_i \geq r_1, \sum r_i = 1\right\}, \\
\mathcal{X}_\mathcal{C} &= \left\{x \in \mathbb{R}_+^n \mid \sum_{i=1}^{n} x_i \leq k\right\}, \\
\mathcal{X}_\mathcal{R} &= \{(0, \ldots, 0)\}, \\
\Delta_\mathcal{C} &= \{I_{n\times n}\}, \\
\Delta_\mathcal{R} &= \Delta_G, \\
F &= \left\lceil \frac{n}{k} \right\rceil, \\
\mathit{Last} &= \mathcal{C}, \\
\mathcal{W}_\mathcal{C} &= \left\{(c, r) \in \mathbb{R}_+^{2n} \mid \forall i \in V(G), c_i = 1\right\}.
\end{aligned}$$

The set $\mathcal{I}$ guarantees that the only possible initial state is the one where the initial vertex is completely marked and the surfer is entirely contained in it. The surfer does

not win the game until it is able to escape the marked area, hence $\mathcal{V}$ shows that the game is not yet lost for the observer while the amount of marks on each vertex is at least the amount of surfer on the same vertex. The sets $\mathcal{X}_\mathcal{C}$ and $\Delta_\mathcal{C}$ guarantees that the observer may not move its marks along edges of the graph. While the sets $\mathcal{X}_\mathcal{R}$ and $\Delta_\mathcal{R}$ guarantees that the surfer moves by sliding, or splitting and sliding, itself along edges of the graph. Since at each step which is not the last the observer might mark at most k vertices, we have that the observer, if it wins the game at all, wins the game in at most $\lceil n/k \rceil$ rounds. Finally, $\mathcal{W}_\mathcal{C}$ states that the observer only wins if it marks all vertices of the graph.

For the next theorem, let $\text{fsn}(G, v)$ be the minimum k such that the observer has a winning strategy in the *fractional Surveillance game*.

Theorem 57. *If $\mathcal{C}$, the observer, wins the* fractional Surveillance game *with k marks in an n-node graph, then $\mathcal{C}$ wins the* Surveillance game *with high probability if it is allowed to use $O(k \log n)$ marks.*

In other words, if $\mathcal{C}$ wins the fractional Surveillance game *against a surfer following a random walk with k marks, then it has a high probability of winning against an integral surfer following the same random walk with $O(k \log n)$ marks.*

Proof. The proof of Theorem 57 closely follows that of the $\log n$ approximation for set cover in [Vaz04].

Assume that $\mathcal{C}$, the observer, and $\mathcal{R}$, the surfer, play the integral *Surveillance game* on a graph G, that $\text{fsn}(G, v) \leq k$, that $V = \{1, \ldots, n\}$ and that the initial vertex is vertex 1. The initial state of the game is (c', r') such that: if $i \neq 1$ then $c'_i = r'_i = 0$ and if $i = 1$ then $c_i = r_i = 1$. Since, from Section 7.3, we have that the number of marks necessary for the observer does not change by restricting the surfer to play in an integral manner, assume, moreover, that the surfer moves in an integral manner. That is, in order to move, the surfer chooses a matrix in $\delta \in \Delta_G \cap \mathbb{N}^n$. Since the initial state of the game we have the surfer entirely on vertex 1, this guarantees that the surfer remains integral during all the game.

In the following we describe the strategy of the observer. Let (c, r) be the current state of the game, which is (c', r') on the first turn of the observer. During each turn t of the observer, let the vector $x^t \in \mathcal{X}_\mathcal{C}$ be the the amount of marks used by the observer, in the *fractional Surveillance game*, when the initial state is given by (c, v). That is, $x^t = (x_1, \ldots, x_n)$ is the amount of marks the observer would place on the vertices of G in order to win against the surfer in the *fractional Surveillance Game*. Then, in the integral game, the observer marks a vertex i if among $O(\log n)$ independent random tests with probability x_i at least one of them is a success.

We want to measure the probability that the observer loses, using this strategy, against any strategy for the surfer in the integral game. Let A_i^t be the event that $r_i > c_i$ at step t of the game. In other words, A_i^t is the event that the observer has lost to the surfer because of vertex i at step t.

Then, $P(A_i^t) \leq (x_i^1)^{c \log n}(x_i^2)^{c \log n} \cdots (x_i^t)^{c \log n}$. Since $\text{fsn}(G, v) \leq k$ we have that $\sum_{i=1}^t x_i^t = 1$. Therefore, from a simple calculus manipulation, $P(A_i^t)$ is minimum when $x_i^1 = x_i^2 = \cdots = x_i^t = 1/t$. Hence, $P(A_i^t) \leq (\frac{1}{t})^{tc \log n} \leq (\frac{1}{e})^{c \log n}$, where e is the base of the natural logarithm.

Then, the probability that the observer loses the game is given by $P(\bigcup_{t=1}^F \bigcup_{i=1}^n A_i^t)$. Therefore, $P(\bigcup_{t=1}^F \bigcup_{i=1}^n A_i^t) \leq n^2(\frac{1}{e})^{c \log n}$. Let $c \geq 3$, then $P(\bigcup_{t=1}^F \bigcup_{i=1}^n A_i^t) \leq \frac{1}{n}$. Therefore, the observer wins the game with high probability.

Moreover, the expected cost of this strategy is given by $\text{fsn}(G, v)c \log n = O(k \log n)$. □

One question that remains open is how far can the gap between the fractional surveillance number and the "integral" surveillance number be.

Angel problem

In its original definition, the *Angel problem* game is played on an infinite grid. While it is possible to define vectors of infinite dimension in order to model configurations of this game, the results in this chapter do not apply in the sense that the algorithms might never finish. Hence, instead of considering the game as being played on an infinite grid we focus on it being played in any finite graph. Then, since at each turn the devil is allowed to mark, or "eat", vertices of the graph, the whole graph will be marked at some point in the game. This means that the angel is doomed to lose eventually. On the other hand, if we give a reasonable limit on the number of turns the game is allowed to last, then it does not seem obvious any more who wins.

Given a graph G, let Δ_G^a be the set of matrices that can be obtained by multiplying any a-tuple of matrices in Δ_G and let $N^s(i)$ be the set of vertices of $V(G) \setminus \{i\}$ that are at distance at most s from i. For example, $\Delta_G^2 = \{\delta_1\delta_2 \mid \delta_1 \in \Delta_G, \delta_2 \in \Delta_G\}$.

The *Angel problem* game where a devil that can mark, or "eat", at most k vertices and an angel that can move along at most s edges at each turn can be modeled with the following sets, we assume that the game is played on a graph $G = (V, E)$ with $V = \{1, \ldots, n\}$ and that the initial vertex is vertex 1:

$$\begin{aligned}
\mathcal{I} &= \{(c_1, \ldots, c_n, r_1, \ldots, r_n) \mid c_1 = 0, r_1 = 1, \forall i \in V(G) \setminus \{1\}, c_i = 0, r_i = 0\},\\
\mathcal{V} &= \left\{(c, r) \in \mathbb{R}_+^{2n} \mid \sum r_i = 1\right\},\\
\mathcal{X}_\mathcal{C} &= \left\{x \in \mathbb{R}_+^n \mid \sum_{i=1}^{n} x_i \leq k\right\},\\
\mathcal{X}_\mathcal{R} &= \{(0, \ldots, 0)\},\\
\Delta_\mathcal{C} &= \{I_{n\times n}\},\\
\Delta_\mathcal{R} &= \Delta_G^s,\\
F &= ?,\\
\mathit{Last} &= \mathcal{R},\\
\mathcal{W}_\mathcal{C} &= \left\{(c, r) \in \mathbb{R}_+^{2n} \mid \forall i \in V(G), \sum_{j \in N^s(i)} c_j \geq r_i\right\}.
\end{aligned}$$

One of the main questions when modeling this game is how big should F, the maximum number of turns, be. Clearly, if $F \geq k/n$, then the whole graph is marked at turn F and the angel is doomed to lose. Therefore, we can assume that F is at most k/n. A smaller value of F might make the game harder for the devil, since it has a smaller amount of turns to "eat" the angel.

The proof of Theorem 58 is very similar with the proof of Theorem 57, since both games have a similar set of rules. The main difference being how the evader, the angel and the surfer, wins the game. The angel start in an unmarked area and tries to stay out of it, while the surfer starts in a marked area and wants to leave it. Let $\text{fang}(G, v)$ be the minimum number of marks such that the devil has a winning strategy in the *fractional Angel problem* game.

Theorem 58. *If $\mathcal{C}$, the devil, wins the* fractional Angel problem *game with k marks in an n-node graph, then the devil wins the* Angel problem *game with high probability if it is allowed to use $O(k \log n)$ marks.*

In other words, if the devil wins the fractional Angel problem *against an angel following a random walk with k marks, then it has a high probability of winning against an integral angel following the same random walk with $O(k \log n)$ marks.*

Proof. Assume that $\mathcal{C}$, the devil, and $\mathcal{R}$, the angel, play the integral *Angel problem* game on a graph G, that $\text{fang}(G, v) \leq k$, that $V = \{1, \ldots, n\}$ and that the initial vertex is vertex 1. The initial state of the game is (c', r') such that: if $i \neq 1$ then $c'_i = r'_i = 0$ and if $i = 1$ then $c_i = 0$ and $r_i = 1$. Since, from Section 7.3, we have that the number of marks necessary for the devil does not change by restricting the angel to play in an integral manner, assume, moreover, that the angel moves in an integral manner. That is, in order to move, the angel chooses a matrix in $\delta \in \Delta^s_G \cap \mathbb{N}^n$. Since the initial state of the game we have the angel entirely on vertex 1, this guarantees that the angel remains integral during all the game.

In the following we describe the strategy of the devil. Let (c, r) be the current state of the game, which is (c', r') on the first turn of the devil. During each turn t of the devil, let the vector $x^t \in \mathcal{X}_\mathcal{C}$ be the the amount of marks used by the devil, in the *fractional Angel problem*, when the initial state is given by (c, v). That is, $x^t = (x_1, \ldots, x_n)$ is the amount of marks the devil would place on the vertices of G in order to win against the angel in the *fractional Angel problem*. Then, in the integral game, the devil marks a vertex i if among $O(\log n)$ independent random tests with probability x_i at least one of them is a success.

We want to measure the probability that the devil does not win at step t, using this strategy, against any strategy for the angel in the integral game. Let $A^t_{i,j}$ be the event that, there is $j \in N^a(i)$ such that $r_i > c_j$ at step t of the game. In other words, $A^t_{i,j}$ is the event that the devil does not win against the angel because of the amount of angel at vertex i at step t.

Then, $P(A^t_{i,j}) \leq \sum_{j \in N^a(i)} (x^1_j)^{c \log n} (x^2_j)^{c \log n} \cdots (x^t_j)^{c \log n}$. Since $\text{fang}(G, v) \leq k$ we have that $\sum_{l=1}^{t} x^l_j = 1$. Therefore, from a simple calculus manipulation, $P(A^t_{i,j})$ is minimum when for all $j \in N^a(i)$ we have $x^1_j = x^2_j = \cdots = x^t_j = 1/t$. Hence, $P(A^t_i) \leq \sum_{j \in N^a(i)} (\frac{1}{t})^{tc \log n} \leq n(\frac{1}{t})^{tc \log n} \leq n(\frac{1}{e})^{c \log n}$, where e is the base of the natural logarithm.

Then, the probability that the devil loses the game is given by $P(\bigcup_{t=1}^{F} \bigcup_{i=1}^{n} A^t_{i,j})$. Therefore, $P(\bigcup_{t=1}^{F} \bigcup_{i=1}^{n} A^t_{i,j}) \leq n^3 (\frac{1}{e})^{c \log n}$. Let $c \geq 4$, then $P(\bigcup_{t=1}^{F} \bigcup_{i=1}^{n} A^t_{i,j}) \leq \frac{1}{n}$. Therefore, the devil wins the game with high probability.

Moreover, the expected cost of this strategy is given by $\text{fang}(G, v) c \log n = O(k \log n)$. □

7.5 Conclusion

In this chapter, we studied a framework that models several turn-by-turn pursuit-evasion games. This framework allows us to naturally define fractional versions of these games. Moreover, we described an algorithm based on linear programming techniques that decides if the pursuer has a winning strategy for any game that can be modeled by this framework.

One disadvantage of our algorithm is that its complexity is more than exponential[2]. However, from Lemma 50 and Lemma 52, the only obstacle to have a polynomial time algorithm in t and n for deciding if player $\mathcal{C}$ has a winning strategy is the complexity of projecting the system of linear inequalities R obtained in Lemma 50 into the $2n$ variables representing the correspondent set $\mathcal{C}_i$. Regrettably, the method we used, the Fourier–Motzkin, for this projection, does not have a running time that is polynomial on the number of linear inequalities of the input. We wonder if, with a finer analysis of the process of projecting or with a different method for projecting, we might decrease this complexity. In other words, the complexity of the decision problem associated with fractional turn-by-turn games is still open.

Nevertheless, while, potentially, the number of constraints can grow more than exponentially in the size of the input graph when computing projection of sets $\mathcal{C}_i$ (Lemma 50), there seems to be several redundant constraints in this projection. For this reason, we are interested in measuring the performance of this algorithm in practice.

One advantage of our algorithm is that the system of linear inequalities constructed by it can be seen as linear relaxations of the "integral" versions of these games. Hence, it is possible to solve "directly" the "integral" version by enforcing the variables to be integrals.

Although the proof in this chapter are restrict to games where both players are allowed to slide and add tokens, the results also hold for games in which:

- players can not slide tokens ($\Delta_G = \{I_{n\times n}\}$);
- Δ_G is different for each player (that is, one player may slide tokens along edges, while the other may not);
- several types of tokens for each player, or several cops/several robbers;
- both players can move more than once in their turn (for example, both cops and robbers with speed $s > 1$ in the *Cops and Robbers* game);
- tokens are on edges instead of vertices.

Despite of the fact that we were only able to prove some (non polynomial) approximation results for the *Surveillance game* and the *Angel problem* game, we think that such approach to turn-by-turn pursuit-evasion games can lead to approximation algorithms for other games compatible with this framework.

On the subject of *Cops and Robbers*, unfortunately, this approach does not seem to be helpful, since for every graph, $1 + \epsilon$, $\epsilon > 0$, cops are enough to capture the robber in a linear number of steps. This is done by a strategy that uniformly spread the cops over the vertices of the graph, ensuring that at least a small amount of cops is on the same vertex as the robber and that this amount is able to follow it. One drawback of this strategy is that it does not work for a robber of speed two. Since a robber of speed two can move along two edges during its turn, a small amount of cops cannot "follow" the robber. It might be interesting to investigate the *fractional Cops and Robbers* when the robber has speed bigger than 1. We hope that this might lead to new insights into solving the Meyniel's conjecture ($O(\sqrt{n})$ cops can capture one robber in any n-node graph, see Chapter 5) for the *Cops and Robbers* game when the robber has speed s .

We finish this part with the question whether this approach can be used for graph searching games.

[2]A function $f(x)$ is more than exponential in x, if $f(x) \neq O(k^x)$ for all $k > 0$.

Part III

Convexity

Chapter 8

Convexity in Graphs

In an effort to extend the concept of convexity, normally associated with Euclidean spaces, Farber and Jamison proposed the following general definition of convexity [FJ86]. An *alignment* over finite set X is a family $\mathcal{C}$ of subsets of X that is closed under intersection and that contains both X and the empty set. The members of $\mathcal{C}$ are called the *convex sets* of X. The pair $(X, \mathcal{C})$ is then called an aligned space. For any $S \subseteq X$, the *convex hull* of S is the smallest member of $\mathcal{C}$ containing S. For any $S \in \mathcal{C}$, a *hull set* of S is a set $S' \subseteq S$ such that S is the convex hull of S'.

The notion of convexity can be extended to graphs and more precisely to their set of vertices, as a specific alignment over it. Since the notion of convexity on a graph depends on the specific alignment over the set of vertices of the graph, there are several different parameters corresponding to different possible alignments. In this chapter, we survey some of the most important results concerning convexity in graphs. In Section 8.1, we show how different alignments over the set of vertices of a graph translates to different convexity measures. We focus on three types of convexity, the Geodesic Convexity, the Monophonic Convexity and the P_3 Convexity, which arguably translates the notion of a "straight line" from Euclidean spaces in a more natural manner. Then, in Section 8.2, we present some of the most important results concerning the hardness of computing parameters associated with these convexities. Finally, in Section 8.2, we survey some structural properties of these three types of convexities.

8.1 Alignments - Types of Convexity

In this section, we define the graph convexities that are explored in the rest of this chapter. Through the rest of this section, unless otherwise stated, let $G = (V, E)$ be any simple and connected graph of order n.

Geodesic Convexity

We start by defining the *Geodesic Convexity* which is related to shortest paths. Roughly, the concept of a straight line is translated as a shortest path.

Let the closed interval, or geodesic, $I[u, v]$ be the set of vertices of G that lie on a shortest path between $u \in V$ and $v \in V$. For any set $S \subseteq V$, let $I[S] = \bigcup_{\{u,v\} \subseteq S} I[u, v]$.

In the *geodesic convexity* [FJ86], a set of vertices S is convex if, for all $\{u, v\} \subseteq S$ such that $w \in I[u, v]$, then $w \in S$. In other words, S is convex if, and only if, $I[S] = S$. Let

the *geodetic number*, geo(G), of G be the minimum integer k such that there exists $S \subseteq V$ with $I[S] = V$ and $|S| \leq k$. Trivial examples of sets that are geodesically convex are the empty set and V. The *(geodetic) convexity number*, $\text{con}_{\text{geo}}(G)$, of a graph $G = (V, E)$, defined in [CWZ02], is the maximum integer k such that there exists a convex set $S \subset V$ with $|S| \leq k$. That is, k is the maximum cardinality of a convex set that is a proper subset of V.

The *geodetic convex hull*, $I_h[S]$, of a set $S \subseteq V$ is the smallest S' such that $S \subseteq S'$ and S' is convex. A *geodetic hull set*, or simply hull set, $S \subseteq V$ is such that $I_h[S] = V$. The *geodetic hull number*, or simply the hull number, hn(G), of G is the minimum $k \geq 0$ such that there is a hull set S with $|S| \leq k$. Figure 8.1 shows an example of a hull set of a graph G.

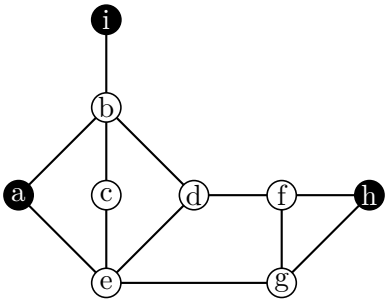


Figure 8.1: Vertices in black represent a hull set of the graph.

As stated in Chapter 1. Given a graph G and $S \subseteq V(G)$, the process to obtain $I_h[S]$ can be seen as an iterative process in the following manner. Starting with $S' = S$ we repeat the following until no more vertices can be added to S'. For every $\{x, y\} \subseteq S'$, we add to S' the vertices in $I[x, y]$. Then, when no more vertices can be added to S' we have that $I_h[S] = S'$. Moreover, S is a hull set of G if, and only if, $S' = V(G)$. The hull number of G is the minimum cardinality S such that $S' = V(G)$, where S' is obtained with this process when applied to S.

Monophonic Convexity

If induced paths are used in place of shortest paths in the definition of $I[u, v]$, we get the *monophonic convexity*. Formally, let the monophonic interval $J[u, v]$ be the set of vertices of G that lie on an induced path between $u \in V$ and $v \in V$. For any set $S \subseteq V$, let $J[S] = \bigcup_{\{u,v\} \subseteq S} J[u, v]$.

In the *monophonic convexity* [FJ86], a set of vertices S is monophonic convex, or m-convex, if, for all $\{u, v\} \subseteq S$ such that $w \in J[u, v]$, then $w \in S$. In other words, S is m-convex if, and only if, $J[S] = S$.

Let the *monophonic number*, mon(G), of G be the minimum integer k such that there exists $S \subseteq V$ with $J[S] = V$ and $|S| \leq k$. The *m-convexity number*, $\text{con}_{\text{mon}}(G)$, of G is the maximum integer k such that there exists a m-convex set $S \subset V$ with $|S| \leq k$.

The *monophonic convex hull*, or simply the m-convex hull, $J_h[S]$ of a set $S \subseteq V$ is the smallest S' such that $S \subseteq S'$ and S' is m-convex. A *monophonic hull set*, or simply m-hull set, $S \subseteq V$ is such that $J_h[S] = V$. The *monophonic hull number*, or simply the m-hull number, mn(G), of G is the minimum $k \geq 0$ such that there is a m-hull set S with $|S| \leq k$. Figure 8.2 shows an example of a m-hull set of a graph G.

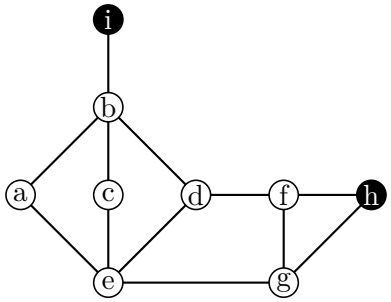


Figure 8.2: Vertices in black represent a m-hull set of the graph.

P_3 Convexity

The last convexity we define is the P_3 Convexity. This convexity is related to paths of length 2 in a similar way that the geodesic convexity is related to shortest paths.

Let the P_3-interval, $I_3[u, v]$, between vertices u and v be defined as $N[u] \cap N[v]$ if $u \neq v$, otherwise $I_3[u, v] = \{u\}$. In other words, $I_3[u, v]$ is the set of vertices of G that lie on any path of length exactly two between $u \in V$ and $v \in V$. For any set $S \subseteq V$, let $I_3[S] = \bigcup_{\{u,v\} \subseteq S} I_3[u, v]$.

In the P_3 *Convexity* [CDS09], a set of vertices S is P_3-convex if, for all $\{u, v\} \subseteq S$ such that $w \in I_3[u, v]$, then $w \in S$. In other words, S is convex if, and only if, $I_3[S] = S$. Let the P_3-*geodesic number*, $\text{geo}_3(G)$, of G be the minimum integer k such that there exists $S \subseteq V$ with $I_3[S] = V$ and $|S| \leq k$. The P_3-*convexity number*, $\text{con}_{\text{geo}_3}(G)$, of G is the maximum integer k such that there exists a P_3-convex set $S \subset V$ with $|S| \leq k$.

The P_3-*convex hull* $I_{3h}[S]$ of a set $S \subseteq V$ is the smallest S' such that $S \subseteq S'$ and S' is P_3-convex. A P_3-*hull set*, $S \subseteq V$ is such that $I_{3h}[S] = V$. The P_3-*hull number*, $\text{hn}_3(G)$, of G is the minimum $k \geq 0$ such that there is a P_3-hull set S with $|S| \leq k$. Figure 8.3 shows an example of a P_3-hull set of a graph G.

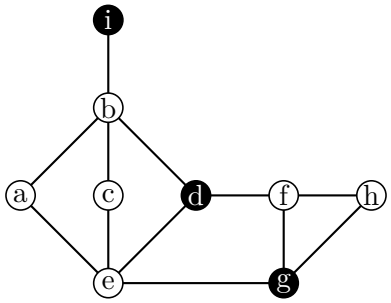


Figure 8.3: Vertices in black represent a P_3-hull set of the graph.

8.2 Algorithmic Aspect of Convexity

In this section, we explore the hardness of decision problems related to the aforementioned convexity parameters.

Geodesic Convexity

The difficulty of computing the geodetic number of a graph was first studied in [HLT93]. It was shown that deciding if the geodetic number of a graph is at most an integer k is NP-complete.

In [DGK+09] Dourado *et al.* proved that deciding whether $\text{hn}(G) \leq k$ for an integer k and a general graph G is NP-complete by a reduction from the 3-SAT problem. They also proved that if G is a unit interval graph, a cograph or a split graph, then this problem can be solved in time polynomial on the number of vertices of the graph.

In particular, the algorithm for computing the hull number of a cograph proposed in [DGK+09] takes advantage of the fact that a modular decomposition of a cograph is composed only of serial or parallel nodes. This is interesting because there are superclasses of cographs that also have a "well behaved" modular decomposition such as P_4-sparse graphs for example. Hence, we expect that the problem of computing the hull number on these classes of graphs to be solvable in polynomial time; something that will be further explored on Chapter 9.

Recently, in [KN13], it was shown that there is a polynomial time algorithm to compute $\text{hn}(G)$ if G is a distance hereditary graph or a chordal graph. The algorithm proposed for computing the hull number of a distance hereditary graph runs in linear time and it is also able to compute the geodetic number of said graph. The main building block of this algorithm is the fact that the split decomposition of distance hereditary graphs is such that each of its blocks is either a clique or a star.

If techniques for computing the hull number can also be used to compute the geodetic number of distance hereditary graphs, can the techniques to compute the hull number of chordal graphs be used to help computing the geodetic number of chordal graphs? This seems unlikely, since, in [DPRS10b], it was shown that deciding if $\text{geo}(G) \leq k$, for a integer k and a chordal graph G, is NP-complete.

For a general graph G and integer k, the problem of deciding if $\text{con}_{\text{geo}}(G) \leq k$ is NP-complete [Gim03]. Dourado *et al.* further improved this result in [DPRS12] by showing that the result still holds even if G is a bipartite graph. A linear time algorithm to compute $\text{con}_{\text{geo}}(G)$ for cographs G was proposed in [DPRS12].

Monophonic Convexity

If, on the one hand, the problem of computing the hull number of graphs is hard, on the other hand, computing $\text{mn}(G)$ for an arbitrary graph $G = (V, E)$ can be done in time $O(|V|^3|E|)$ [DPS10]. The algorithm proposed by Dourado *et al.* starts by computing a *clique decomposition tree* [Tar85] of the graph. Then, the monophonic hull number of the graph can be obtained by, roughly, counting the number of its simplicial vertices and the number of "special" leaves of this decomposition.

Note that, for any distance hereditary graph G and for any $\{x, y\} \subseteq V(G)$, $J[u, v] = I[u, v]$. Hence, for any distance hereditary graph, we have that $\text{hn}(G) = \text{mn}(G)$, $\text{mon}(G) = \text{geo}(G)$ and $\text{con}_{\text{geo}}(G) = \text{con}_{\text{mon}}(G)$. Therefore, since in distance hereditary graphs any induced path is also a shortest path, the previous algorithm to compute the m-hull number of distance hereditary graphs also computes their hull number. Another consequence of this relation is that $\text{mon}(G)$ and $\text{mn}(G)$ can be computed in linear time for distance hereditary graphs from the previous mentioned result in [KN13].

Dourado *et al.* showed in [DPS10] that the decision problems associated with the monophonic number and the m-convexity number are NP-Complete for arbitrary graphs.

P_3 Convexity

The problem of deciding if $\text{hn}_3(G) \leq k$ is NP-complete for a general graph G and integer k [CDP+11]. A relationship resembling the one between monophonic convexity and

Table 8.1: Table comparing complexity of problems associated with monophonic and geodesic convexity on a graph $G = (V, E)$ with $|V| = n$ and $|E| = m$. The set S denotes any subset of V. u and v are vertices of V. The "X" can correspond to either Geodetic, Monophonic or P_3-convexity depending on the column of the table.

Problem	Geodetic	Monophonic	P_3-Convexity
Is S convex?	$O(nm)$ [DGK$^+$09]	$O(nm)$ [DPS10]	$O(n+m)$[1]
Computing $I[u,v]$	$O(m)$ [DGK$^+$09]	NP-complete [DPS10]	$O(n)$[2]
Convex Hull of S	$O(nm)$ [DGK$^+$09]	$O(n^2m)$ [DPS10]	polynomial [DJR09]
X Convexity Number	NP-complete [DPRS10b]	NP-complete [DPS10]	NP-complete [CDD$^+$10]
X Number	NP-complete [DPRS12]	NP-complete [DPS10]	NP-complete [JP89]
X Hull Number	NP-complete [DGK$^+$09]	$O(n^3m)$ [DPS10]	NP-complete [CDP$^+$11]

geodesic convexity on distance hereditary graphs exists between the P_3 convexity and geodesic convexity. A simple consequence of a result in [SS13] is that the concepts of P_3 convexity and geodesic convexity coincide in graphs that are distance hereditary and have an universal vertex. If G is a distance hereditary graph with an universal vertex w, then $I_3[u,v] = I[u,v] \cup \{w\}$ for any $u, v \in V(G)$. Therefore, complexity results for distance hereditary graphs with an universal vertex are valid for all the aforementioned convexities.

Deciding if $\text{geo}_3(G) \leq k$, for some integer k, in a general graph G is also NP-complete. In order to see this, consider the following relationship with the 2-domination problem. For any graph $G = (V, E)$ and a set $S \subseteq V$, we have that S is a 2-dominating set[1] of G if and only if S is P_3-convex and $I_3[S] = V$. In other words, the complexity of deciding if $\text{geo}_3(G) \leq k$ for some integer k is the same as the complexity of deciding if there is a set S such that $|S| \leq k$ and S is a 2-dominating set of V. The result follows from the fact that the minimum 2-dominating set problem is NP-complete [JP89]. However, when G is restricted to be a tree or a cograph, then computing $\text{geo}_3(G)$ can be done in polynomial time [CDS09].

In [CDD$^+$10], it was shown that it is NP-complete to decide for a split graph G and integer k, if $\text{con}_{\text{geo}_3}(G) \leq k$. However, similarly to the P_3-geodetic number, when G is restricted to be a tree or a cograph, then computing $\text{con}_{\text{geo}_3}(G)$ can be done in polynomial time [CDS09].

Table 8.1 summarizes some of the results presented in this section, by showing the difficulty of problems related to geodesic, monophonic and P_3 convexities on general graphs.

8.3 Structural Aspect of Convexity

Since the problem of computing the aforementioned convexity parameters is often hard, there is an interest in finding bounds for these parameters in order to better understand them.

[1]Given a graph G, a set $S \subseteq V(G)$ is k-dominating if, for every vertex $v \in V(G) \setminus S$, $|N(v) \cap S| \geq k$.

[1]A simple search for any vertex that is not on S, nor have two neighbors in S suffices.

[2]This can be done by taking $N[u] \cap N[v]$, since $I_3[u,v] = N[u] \cap N[v]$.

Bounding the Convexity

Let $\mathrm{diam}(G)$ be the diameter,the length of the longest shortest path, of G. Several bounds for the hull number of graphs were shown in [DPRS10a]. If G is a connected triangle-free on n vertices such that $\delta(G) \geq 3$ and $\mathrm{diam}(G) \geq 4$, then $\mathrm{hn}(G) \leq \frac{n-\mathrm{diam}(G)+3}{3}$. If we restrict G to be a connected cubic[3] triangle-free graph on n vertices and $\mathrm{diam}(G) \geq 4$, then $\mathrm{hn}(G) \leq 2\frac{n-\mathrm{diam}(G)+5}{7}$. For any connected graph G on n vertices such that $\mathrm{girth}(G) \geq 5$ and $\delta(G) \geq 2$, then $\mathrm{hn}(G) \leq 2 + \frac{n-\mathrm{diam}(G)-1}{\lceil(\mathrm{girth}(G)-1)/2\rceil}$.

The idea behind how these bounds can be obtained on a graph G is, roughly, the following. Starting with $S = S' = \emptyset$. S is used to represent a hull set, while S' is used to represent $I_h[S]$. Then, while $S' \neq V(G)$ we do the following. If $S' \neq I[S']$ we add to S' all vertices in $I[S']$. If $S' = I[S']$, then all vertices that can be "generated" by vertices in S are already in S', in other words, S' is convex. In this case, we need to choose another vertex to put in S in hopes that $I_h[S] = V(G)$. Then, we greedily choose the vertex of $V(G) \setminus S'$ which is farther away than any vertex in S' and add this vertex to both S and S'. If after its addition to S, $|I_h[S] \setminus S'| \geq k$, this means that, by adding one vertex to S, we are able to "generate" k other vertices. Moreover, if we are always able to "generate" k other vertices when adding a vertex to S, then $\mathrm{hn}(G) \leq n/k$.

If G is a connected unit interval graph with s simplicial vertices, then $\mathrm{geo}(G) \leq s + 2(\mathrm{diam}(G) - 1)$. If G is a triangle-free graph such that $\delta(G) \geq 2$, then $\mathrm{geo}(G) \leq 2|M|$ where M is a maximal matching of G. Furthermore, the geodetic number of chordal and split graphs were fully characterized in [DPRS10b].

A first result related to the (geodetic) convexity number of a graph is due to Chartrand *et al.*. It was shown in [CWZ02] that for any graph G of order n we have that $\mathrm{con}_{\mathrm{geo}}(G) = n - 1$ if and only if G has a simplicial vertex.

However, if G does not have a simplicial vertex, how small can the (geodetic) convexity number be? This question was answered in [Kim04]. Kim showed that for every pair of integers k and n with $2 \leq k \leq n - 1$, there exists a connected triangle-free graph of order n with $\mathrm{con}_{\mathrm{geo}}(G) = k$. In [Kim04], Kim also proved a first upper bound for the (geodetic) convexity number of k-regular graphs. If $3 \geq k + 1 < n$ and G is a k-regular graph of order n, then $\mathrm{con}_{\mathrm{geo}}(G) \geq \frac{n}{n-k}$.

Products of Graphs and Convexity

In this section, we overview the known results related to convexity parameters of (lexicographic, Cartesian or strong) products of graphs. Understanding how convexity parameters behave under a product operation might allows us to expand the convexity results for some graph classes into results for graphs that are obtained through the product of graphs in these classes.

In order to present these results, we first define each product operation. Let G and H be two graphs.

The *lexicographic product*, $G \odot H$, of G and H is the graph whose vertex set is $V(G \odot H) = V(G) \times V(H)$ and such that two vertices (g_1, h_1) and (g_2, h_2) are adjacent if, and only if, either $\{g_1, g_2\} \in E(G)$ or we have that both $g_1 = g_2$ and $\{h_1, h_2\} \in E(G)$. For a vertex $g \in V(G)$, let its *H-layer* in $G \odot H$ be the set $H(g) = \{(g, h) \in V(G \odot H) \mid h \in V(H)\}$.

[3] A cubic graph is such that every vertex has degree three.

The *Cartesian product*, $G \times H$, of G and H is the graph whose vertex set is $V(G \times H) = V(G) \times V(H)$ and such that two vertices (g_1, h_1) and (g_2, h_2) are adjacent if, and only if, either $g_1 = g_2$ and $\{h_1, h_2\} \in E(H)$; or $h_1 = h_2$ and $\{g_1, g_2\} \in E(G)$.

Finally, the *strong product*, $G \boxtimes H$, of G and H is the graph whose vertex set is $V(G \boxtimes H) = V(G) \times V(H)$ and such that two vertices (g_1, h_1) and (g_2, h_2) are adjacent if, and only if, $\{g_1, g_2\} \in E(G)$ or $g_1 = g_2$; and $\{h_1, h_2\} \in E(H)$ or $h_1 = h_2$.

In [CJG02], it was proved that if G is a connected graph of order n and $m \geq 1$, then $\mathrm{con}_{\mathrm{geo}}(G \odot K_m) = mn - 1$ if, and only if, $1 \leq n \leq 2$; or $n \geq 3$ and G has a simplicial vertex. Canoy Jr. and Garces also showed that the geodesic convexity number of the Cartesian product of two graphs can be fully characterized with the following theorem.

Theorem 59 ([CJG02])**.** *If G and H are two connected graphs, then:*

$$\mathrm{con}_{\mathrm{geo}}(G \times H) = \max\{|V(G)|\,\mathrm{con}_{\mathrm{geo}}(H), |V(H)|\,\mathrm{con}_{\mathrm{geo}}(G)\}.$$

In [CCJ04], a full characterization of the hull number of Cartesian products of graphs was proposed.

Theorem 60 ([CCJ04])**.** *If G and H are two connected graphs, then:*

$$\mathrm{hn}(G \times H) = \max\{\mathrm{hn}(G), \mathrm{hn}(H)\}.$$

Then, in [BKT08], a full characterization of the geodetic number of Cartesian products of graphs was given.

Theorem 61 ([BKT08])**.** *If G and H are two connected graphs, then:*

$$\mathrm{geo}(G \times H) = \max\{\mathrm{geo}(G), \mathrm{geo}(H)\}.$$

Let G and H be graphs with at least two vertices and Y be any proper subset of the vertices of the lexicographic product of G and H with the restriction that Y does not induce a complete graph. In [ACKP12], Anand *et al.* showed a set of conditions that are necessary and sufficient to decide if Y is (geodesic) convex or monophonic convex.

In [CHM$^+$10], Cáceres *et al.* studied the geodetic number and hull number of strong product of graphs. Let G and H be two graphs. A first result shows that if $S_G \subseteq V(G)$ is a hull set of G and $S_H \subseteq V(H)$ a hull set of H then $S_G \times S_H$ is a hull set of $G \boxtimes H$. The same result is true if S_G and S_H are *geodetic sets* of G and H respectively, that is, if $I[S_G] = V(G)$ and $I[S_H] = V(H)$. Another result states that:

Theorem 62 ([CHM$^+$10])**.** *For any two graphs G and H we have that:*

$$\min\{\mathrm{geo}(G), \mathrm{geo}(H)\} \leq \mathrm{geo}(G \boxtimes H) \leq \mathrm{geo}(G)\,\mathrm{geo}(H).$$

Furthermore, both bounds are sharp.

Moreover, if G has no simplicial vertices then $\mathrm{hn}(G \boxtimes H) \leq \mathrm{hn}(G)$. This last bound is also sharp.

We refer to [BKT11] and [CMS05] for surveys on path convexities of graphs and related concepts.

8.4 Objectives

In the next chapter, Chapter 9, we aim at further investigating the question of how hard it is to compute the hull number of a graph even when we are restricted to specific classes of graphs. As a product of this investigation, we show that the problem of computing the hull number of an arbitrary graph is in FPT, by proposing FPT-algorithms for it where the parameter is either the neighborhood diversity of or the number of induced paths of length three.

In Chapter 9, we also investigate how some structural properties of graphs could influence their Hull Number. As a result, we could provide new bounds for the hull number of some graph classes and obtain the hull number of the lexicographic product of two graphs based on the hull number of its factors. Although the main technique for obtaining these bounds is not different from the one presented in this chapter, the bounds we have found are not comparable with the ones presented in this chapter.

Chapter 9

On the Hull Number of Graphs

In this chapter, we mainly investigate the complexity of computing the hull number of a graph. We try to understand where the difficulty of this problem lies. In order to do that, we focus on some particular graph classes. We also propose FPT-algorithms for computing the hull number of general graphs, where the parameter can be either the number of its induced P_4's, its vertex cover number, or its neighborhood diversity.

A first preliminary section, Section 9.1, recalls important results and definitions used throughout this chapter. In Section 9.2, we answer an open question of Dourado *et al.* [DGK$^+$09] by showing that the decision problem associated with computing the hull number of a bipartite graph is NP-complete. Then, in Section 9.3, we show a polynomial time algorithm to compute the hull number in complement of bipartite graphs. Section 9.4, we extend the algorithm to compute the hull number of cographs in [DGK$^+$09] to the superclass of $(q, q-4)$-graphs. Section 9.5, is devoted to the study of the hull number of $\{P_5, K_3\}$-free graphs. In Section 9.6, we show some rules that can be applied to the graph in order to reduce its size while its hull number behaves in a controlled manner. With the help of these rules we are able to construct a FPT-algorithm to compute the hull number of a general graph where the fixed parameter can be either the *neighborhood diversity* or the *vertex cover number*. We also show how these rules can also be employed to characterize the hull number of lexicographic product of graphs. Section 9.7, is dedicated to the study of the behavior of the hull number of a graph by means of the hull number of its two connected components. This allows us to design a polynomial time algorithm to compute the hull number of cacti graphs. Finally, in Section 9.8, we prove tight upper bounds on the hull number some graphs.

9.1 Terminology and Tools

Otherwise stated, all graphs considered in this chapter are simple, undirected and connected. Let $G = (V, E)$ be a graph. A subgraph H of G is *isometric* if, for any $u, v \in V(H)$, the distance $\text{dist}_H(u, v)$ between u and v in H equals $\text{dist}_G(u, v)$.

In order to avoid unnecessary backtracking we recall definitions of geodesic convexity.

Given a connected graph $G = (V, E)$, the *closed interval* $I[u, v]$ of any two vertices $u, v \in V$ is the set of vertices that belong to some *u-v geodesic* of G, i.e., some shortest (u, v)-path. For any $S \subseteq V$, let $I[S] = \bigcup_{u,v \in S} I[u, v]$. A subset $S \subseteq V$ is *(geodesically) convex* if $I[S] = S$. Given a subset $S \subseteq V$, the *convex hull* $I_h[S]$ of S is the smallest

convex set that contains S.

We say that a vertex v is *generated* by a set of vertices S if $v \in I_h[S]$. Equivalently, given a set S, let $I^0[S] = S$ and $I^k[S] = I[I^{k-1}[S]]$, for $k > 0$. We say that v is generated by S *at step* $t \geq 1$, if $v \in I^t[S]$ and $v \notin I^{t-1}[S]$. Observe that the convex hull $I_h[S]$ of S is equal to $I^{|V(G)|}[S]$. We say that S is a *hull set* of G if $I_h[S] = V$. The size of a minimum hull set of G is the *hull number* of G, denoted by $\mathrm{hn}(G)$ [ES85].

The rest of this section is devoted to basic lemmas on hull sets. These lemmas will serve as cornerstone of most of the results presented in this chapter.

Lemma 63 ([ES85])**.** *For any hull set S of a graph G, S contains all simplicial vertices of G.*

Lemma 63 shows us that a simple lower bound for the hull number of a graph is the number of its simplicial vertices. Simplicial vertices must be on every hull set, since their neighborhood is complete. It occurs because of, the shortest path between any two vertices of the graph does not have any internal vertex that is simplicial.

Lemma 64 ([DGK$^+$09])**.** *Let G be a graph which is not complete. No hull set of G with cardinality $\mathrm{hn}(G)$ contains a universal vertex.*

Lemma 64 allows us to disconsider universal vertices when trying to build a hull set a graph, unless it is complete. Since the only hull set of a complete graph is its set of vertices, this lemma provides us an useful tool to disconsider unnecessary vertices.

Lemma 65 ([DGK$^+$09])**.** *Let G be a graph, H be an isometric subgraph of G and S be any hull set of H. Then, the convex hull of S in G contains $V(H)$.*

Lemma 65 provides us a tool to understand the behavior of convex hulls when considering some particular subgraphs. This, sometimes, allows us to combine hull sets for subgraphs in order to obtain a hull set for the whole graph.

Lemma 66 ([DGK$^+$09])**.** *Let G be a graph and S a proper and non-empty subset of $V(G)$. If $V(G) \setminus S$ is convex, then every hull set of G contains at least one vertex of S.*

Lemma 66 can be seen as a generalization of Lemma 63, since for any simplicial vertex $v \in V(G)$ the set $V(G) \setminus \{v\}$ is convex. The proof and usefulness of Lemma 66 is similar to the one of Lemma 63. We choose to keep both lemmas in order to simplify some of proofs found in this chapter.

9.2 Bipartite Graphs

In this section, we answer an open question of Dourado *et al.* [DGK$^+$09] by showing that the Hull Number Problem is NP-complete for the class of bipartite graphs. Since the Hull Number Problem is in NP, as proved in [DGK$^+$09], it only remains to prove the following theorem:

Theorem 67. *Computing the hull number of a bipartite graph G is NP-hard.*

Proof. To prove this theorem, we adapt the proof presented in [DGK$^+$09]. We reduce the 3-SATisfiability Problem to the problem of computing the hull number of a bipartite graph. Let us consider the following instance of 3-SAT. Given a formula in the conjunctive normal form, let $\mathcal{F} = \{C_1, C_2, \ldots, C_m\}$ be the set of its 3-clauses and $X = \{x_1, x_2, \ldots, x_n\}$

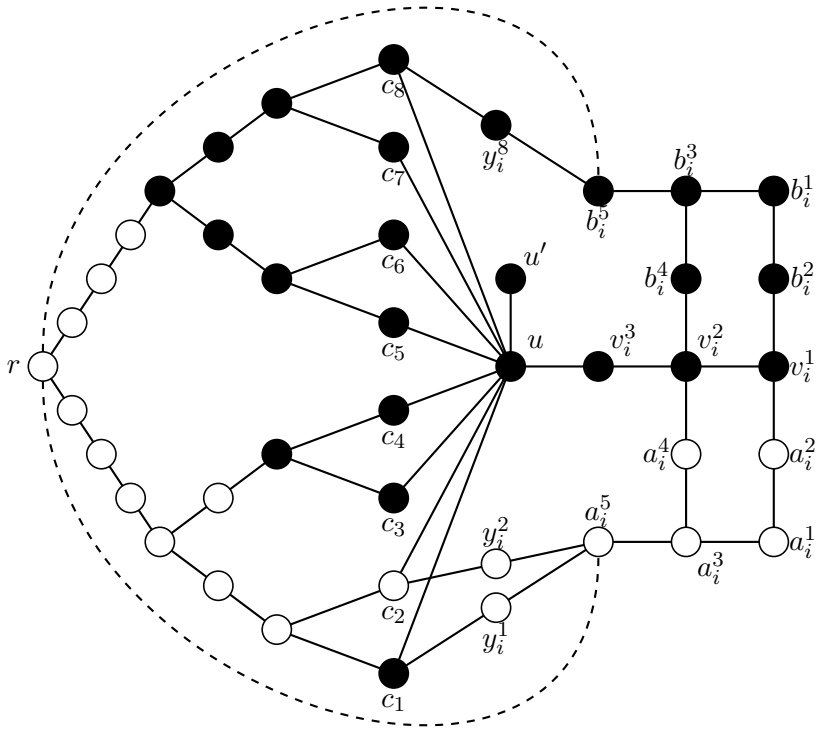


Figure 9.1: Subgraph of the bipartite instance $G(\mathcal{F})$ containing the gadget of a variable x_i that appears positively in clauses C_1 and C_2, and negatively in C_8. If x_i appears positively in C_j, link a_i^5 to c_j through y_i^j. If it appears negatively, we use b_i^5 instead of a_i^5.

the set of its boolean variables. We may assume that $m = 2^p$, for a positive integer $p \geq 1$, since it is possible to add dummy variables and clauses without changing the satisfiability of $\mathcal{F}$ and such that the size of the instance is at most twice the size of the initial instance. Moreover, we also assume, without loss of generality, that each variable x_i and its negation appear at least once in $\mathcal{F}$ (otherwise the clauses where x_i appears can always be satisfied).

Let us construct the bipartite graph $G(\mathcal{F})$ as follows. First, let T be a full binary tree of height p rooted in r with $m = 2^p$ leaves, and let $L = \{c_1, c_2, \ldots, c_m\}$ be the set of leaves of T. We then construct a graph H as follows. First, let us add a vertex u that is adjacent to every vertex in L. Then, any edge $\{w, v\} \in E(T)$ with w the parent of v is replaced by a path with $2^{h(v)}$ edges, where $h(v)$ is the distance between v and any of its descendant leaves. Note that, in H, the distance between r and any leaf is $\sum_{i=0}^{p-1} 2^i = 2^p - 1 = m - 1$. Moreover, it is easy to see that $|V(H)| = O(m \log m)$.

The following claims are proved in [DGK$^+$09].

Claim 18. *Let $v, w \in V(T) \setminus \{r\}$. The closed interval of v, w in H contains the parents of v in T if and only if v and w are siblings in T.*

Claim 19. *The set L is a* minimal *hull set of H.*

Then, let H' be obtained by adding a one degree vertex u' adjacent to u in H. Finally, we build a graph $G(\mathcal{F})$ from H' by adding, for any variable x_i, $i \leq n$, the gadget defined as follows.

Let us start with a cycle $\{a_i^1, a_i^2, v_i^1, b_i^2, b_i^1, b_i^3, b_i^4, v_i^2, a_i^4, a_i^3\}$ plus the edge $\{v_i^2, v_i^1\}$. Then, add the vertex v_i^3 as common neighbor of v_i^2 and u. Add a neighbor b_i^5 (resp., a_i^5) adjacent to b_i^3 (resp., a_i^3) and a path of length $2^{h(r)} - 3 = m - 3$ edges between b_i^5 (resp., a_i^5) and

r. Let D be the set of internal vertices of all these $2n$ paths between a_i^5 (resp., b_i^5) and r, $i \leq n$. Finally, for any clause C_j in which x_i appears, if x_i appears positively (resp., negatively) in C_j then add a common neighbor y_i^j between c_j and a_i^5 (resp., b_i^5). See an example of such a gadget in Figure 9.1. Note that $|V(G(\mathcal{F}))| = O(m(n + \log m))$.

Lemma 68. *$G(\mathcal{F})$ is a bipartite graph.*

Proof. Let us present a proper 2-coloring c of $G(\mathcal{F})$. Let $c(r) = 1$, and for each vertex w in $V(H)$, define $c(w)$ as 1 if w is in an even distance from r, and 2 otherwise. Clearly, c is a partial *proper* coloring of $G(\mathcal{F})$ and moreover we have $c(u) = 1$ and $c(c_j) = 2$, for any $j \in \{1, \ldots, m\}$ (Indeed, any c_i is at distance $m - 1$ (odd) of r in H). Let $c(u') = 2$. For every $i \in \{1, \ldots, n\}$ and for any j such that $x_i \in C_j$, let $c(y_i^j) = 1$. For any $i \leq n$, for any $x \in \{b_i^5, a_i^5, v_i^3, b_i^4, a_i^4, b_i^1, v_i^1, a_i^1\}$, $c(x) = 2$.

Again, this partial coloring of $G(\mathcal{F})$ is proper. One can easily verify that this coloring can be extended to $\{a_i^1, a_i^2, v_i^1, b_i^2, b_i^1, b_i^3, b_i^4, v_i^2, a_i^4, a_i^3\}$ for any $i \leq n$. Moreover, since $c(r) = 1$ and $c(a_i^5) = 2$ ($c(b_i^5) = 2$), for every $i \in \{1, \ldots, n\}$, and since the path that we add in $G(\mathcal{F})$ between r and a_i^5 (b_i^5) is of odd length $m - 3$, one can completely extend c in order to get a proper 2-coloring of $G(\mathcal{F})$. □

Claim 20. *The set $V(G(\mathcal{F})) \setminus \{a_i^1, a_i^2, v_i^1, b_i^1, b_i^2\}$ is convex, for any $i \in \{1, \ldots, n\}$.*

Proof. Denote $W_i = \{a_i^1, a_i^2, v_i^1, b_i^1, b_i^2\}$, for some $i \in \{1, \ldots, n\}$, and $W_i' = \{a_i^3, b_i^3, v_i^2\}$. By contradiction, suppose that there exists an (x, y)-shortest path containing a vertex of W_i, for some $x, y \in V(G(\mathcal{F})) \setminus W_i$. Observe that it implies that that there are $x', y' \in W_i'$ such that $I[x', y']$ contains a vertex of W_i, since W_i' contains all the neighbors of W_i in $V(G(\mathcal{F})) \setminus W_i$. However, it is easy to verify that for any pair $x, y \in W_i'$, $I[x, y]$ contains no vertex of W_i. This is a contradiction. □

Lemma 69. *$hn(G(\mathcal{F})) \geq n + 1$.*

Proof. Let S be any hull set of $G(\mathcal{F})$. Clearly $u' \in S$, because u' is a simplicial vertex of $G(\mathcal{F})$ (Lemma 63). Furthermore, Claim 20 and Lemma 66 imply that S must contain at least one vertex w_i of the set $\{a_i^1, a_i^2, v_i^1, b_i^1, b_i^2\}$, for every $i \in \{1, \ldots, n\}$. Hence, $|S| \geq n + 1$. □

The main part of the proof consists in showing:

Lemma 70. *$\mathcal{F}$ is satisfiable if and only if* $\mathrm{hn}(G(\mathcal{F})) = n + 1$.

First, consider that $\mathcal{F}$ is satisfiable. Given an assignment A that turns $\mathcal{F}$ true, define a set S as follows. For $1 \leq i \leq n$, if x_i is true in A add a_i^1 to S, otherwise add b_i^1 to S. Finally, add u' to S. Note that $|S| = n + 1$. We show that S is a hull set of $G(\mathcal{F})$. First note that $a_i^5, c_j \in I[a_i^1, u']$, for every clause C_j containing the positive literal of x_i. Similarly, observe that $b_i^5, c_j \in I[b_i^1, u']$, for every clause C_j containing the negative literal of x_i. Since A satisfies $\mathcal{F}$, it follows $L \subseteq I_h[S]$. Therefore, H being an isometric subgraph of $G(\mathcal{F})$, Lemma 65 and Claim 20 imply that $V(H) \subseteq I_h[S]$. Furthermore, the shortest paths between r and u have length m, which implies that all vertices a_i^5, b_i^5, y_i^j ($i \leq n$) and all vertices in D are included in $I_h[S]$. It remains to observe that $I_h[a_i^5, b_i^5, w, u']$, where $w \in \{a_i^1, b_i^1\}$, contains the variable subgraph of x_i. Therefore we have that S is a hull set of $G(\mathcal{F})$.

We prove the sufficiency by contradiction. Suppose that $G(\mathcal{F})$ contains a hull set S with $n + 1$ vertices and that $\mathcal{F}$ is not satisfiable.

Recall that, by Lemma 63, $u' \in S$. For any $i \leq n$, let W_i as defined in Claim 20. Recall also that there must be a vertex $w_i \in W_i \cap S$, for any $i \leq n$. Since $v_i^1 \in I[u', a_i^1]$, $v_i^1 \in I[u', b_i^1]$, $a_i^2 \in I[u', a_i^1]$ and $b_i^2 \in I[u', b_i^1]$, we can assume, without loss of generality, that $w_i \in \{a_i^1, b_i^1\}$, for every $i \in \{1, \ldots, n\}$ (indeed, if $w_i \in \{v_i^1, a_i^2\}$, it can be replaced by a_i^1, and if $w_i = b_i^2$, it can be replaced by b_i^1). Therefore S defines the following truth assignment $\mathcal{A}$ to $\mathcal{F}$. If $w_i = a_i^1$ set x_i to true, otherwise set x_i to false. As $\mathcal{F}$ is not satisfiable, there exists at least one clause C_j not satisfied by $\mathcal{A}$.

Using the hypothesis that $\mathcal{F}$ is not satisfiable, we complete the proof by showing that there is a non empty set U such that $V(G(\mathcal{F})) \setminus U$ is a convex set and $U \cap S = \emptyset$. That is, we show that $I_h[S] \subseteq V(G(\mathcal{F})) \setminus U$ for some $U \neq \emptyset$, contradicting the fact that S is a hull set.

For any clause C_j, let us define the subset U_j of vertices as follows. Let P_j be the path in T between c_j and r, let X_j be the p vertices in $V(T) \setminus V(P_j)$ that are adjacent to some vertex in P_j. Then, U_j is the union of the vertices that are either in P_j or that are internal vertices of the paths resulting of the subdivision of the edges $\{x, y\}$ where $x, y \in P_j \cup X_j$. Another way to build the set U_j is to start with the set of vertices in the (unique) shortest path between c_j and r in H and then add successively to this set, the vertices of $V(H) \setminus (V(T) \cup \{u\})$ that are adjacent to some vertex of the current set.

Now, let $U' = \bigcup_{j \in J} U_j$ where J is the (non empty) set of clauses that are not satisfied by $\mathcal{A}$. Note that $r \in U'$.

For any $i \leq n$, let Z_i be defined as follows. If $w_i = a_i^1$ (x_i assigned to true by $\mathcal{A}$), then Z_i is the union of $\{b_i^\ell \mid \ell \leq 5\}$ with the set of the y_i^k that are adjacent to b_i^5. Otherwise, $w_i = b_i^1$ (x_i assigned to false by $\mathcal{A}$), then Z_i is the union of $\{a_i^\ell \mid \ell \leq 5\}$ with the set of the y_i^k that are adjacent to a_i^5.

Finally, let $U = U' \cup (\bigcup_{i \leq n} Z_i) \cup D$. In Figure 9.1, U is depicted by the white vertices, assuming that clause C_2 is false and that x_i is set to false by $\mathcal{A}$. Observe that $U \cap S = \emptyset$.

It remains to prove that $V(G(\mathcal{F})) \setminus U$ is a convex set. Consider the partition $\{A_1, A_2, A_3\}$ of $V(G(\mathcal{F})) \setminus U$ where $A_1 = V(H) \setminus (U \cup \{u\})$, $A_2 = \{u, u'\}$ and $A_3 = V(G(\mathcal{F})) \setminus (U \cup A_1 \cup A_2)$. To prove that $V(G(\mathcal{F})) \setminus U$ is convex, let $w \in A_i$ and $w' \in A_j$ for some $i, j \in \{1, 2, 3\}$. We show that $I[w, w'] \cap U = \emptyset$ considering different cases according to the values of i and j. Recall that $V(H) \setminus \{u\}$ induces a tree T' rooted in r and that, if a vertex of T' is in A_1, then, by definition of U', all its descendants in T' are also in A_1 (i.e., if $v \in U \cap V(T')$, then all ancestors of v in T' are in U). It is important to note that, for any vertex v in A_1, the shortest path in $G(\mathcal{F})$ from v to any leaf ℓ of T' is the path from v to ℓ in T' (in particular, such a shortest path does not pass through r and any vertices in D).

- The case $i = j = 2$, i.e., $m, m' \in \{u, u'\}$, is trivial;

- First, let us assume that $w \in A_1 = V(H) \setminus (U \cup \{u\})$ and $w' \in A_2 = \{u, u'\}$. If $w' = u$ (resp., if $w' = u'$) then $I_h[w, w']$ consists of the subtree of T' rooted in w union u (resp., union u and u'). Hence, $I_h[w, w'] \cap U = \emptyset$ because no descendants of w in T' are in U.

- Second, let $w, w' \in A_1$. If one of them, say w, is an ancestor of the other in T', then $I_h[w, w']$ consists of the path between them in T' (remember that $r \in U$ so $w \neq r$). Since no descendants of w in T' are in U, $I_h[w, w'] \cap U = \emptyset$. Otherwise, there are three cases: (1) either $I_h[w, w']$ consists of the path P between w and w' in T', or (2) $I_h[w, w']$ consists of the union of the subtree R of T' rooted in w, the

subtree R' of T' rooted in w' and u, or (3) $I_h[w,w'] = R \cup R' \cup P \cup \{u\}$. Again, $(R \cup R' \cup \{u\}) \cap U = \emptyset$ because no descendants of w and w' in T' are in U. Hence, it only remains to prove that when $P \subseteq I_h[w,w']$ then $P \cap U = \emptyset$. It is easy to check that $P \subseteq I_h[w,w']$ only in the following case: there exist $x, y, z \in V(T)$ such that x is the parent of y and z in T, and w (resp., w') is a vertex of the path resulting from the subdivision of $\{x,y\}$ (resp., $\{x,z\}$). In this case, it means that all clause-vertices that are descendants of y and z are not in U. Therefore $x \notin U$ and hence no descendants of x are in U. In particular, $P \cap U = \emptyset$.

- Assume now that $w \in A_3$. Let $i \leq n$ such that w belongs to the gadget G_i corresponding to variable x_i. Let us assume that $w_i = b_i^1$. The case $w_i = a_i^1$ can be handled in a similar way by symmetry. Then, by definition, U contains $\{a_i^1, \ldots, a_i^5\}$ and the y_i^j's adjacent to a_i^5. With this setting, x_i is set to false in the assignment $\mathcal{A}$. If there is a vertex y_i^j adjacent to b_i^5, let C_j be the other neighbor of j_i^j. By definition, it means that clause C_j contains the negation of variable x_i. Since x_i is set to false, it means that clause C_j is satisfied and so $C_j \notin U$.

  Let $x \in V(G_i) \setminus U$. Then, any shortest path P from w to x either passes through $V(G_i) \setminus U$ or, there is y_i^j adjacent to b_i^5 such that P passes through y_i^j, C_j, u and v_i^3 (the latter case may occur if $a \in \{y_i^j, b_i^5\}$ and $b = v_i^3$, or $a = y_i^j$ and $b \in \{v_i^3, v_i^2\}$ where $\{a,b\} = \{x,w\}$). Hence, such a path P avoids U, and the result holds if $x = w' \in A_3 \cap G_i$.

  Similarly, if $x \in \{u, u'\}$, then, any shortest path P from w to x either passes through $V(G_i) \setminus U$ or through y_i^j, C_j, u with y_i^j adjacent to b_i^5. In particular, if $x = w' \in \{u, u'\} = A_2$, then the result holds.

  Now, let $x = C_{j'}$ be a leaf of T' that is not in U. Then, any shortest path P from w to x either passes through u or through y_i^j, C_j and, if $j \neq j'$, through u. In any case, P avoids U. If $w' \in A_3 \setminus G_i$, any path between w and w' passes through u or through one or two leaves that are not in U. Finally, if $w' \in A_1$, let R be the subtree of T' rooted in w'. Note, $V(R) \subseteq I_h[w,w']$. Moreover, any shortest path from w to w' contains a leaf of R, i.e., a leaf not in U. By previous remarks, in all these cases, the shortest paths between w and w' avoid u, and $I_h[w,w']$ are disjoint from U.

□

We conclude this section by showing one *approximability result*.

Let $\mathrm{IG}(G)$ be the *incidence graph* of G, obtained from G by subdividing each edge once. That is, let us add one vertex s_{uv}, for each edge $\{u,v\} \in E(G)$, and replace the edge $\{u,v\}$ by the edges $\{u, s_{uv}\}$ and $\{s_{uv}, v\}$.

Proposition 1.

$$\mathrm{hn}(\mathrm{IG}(G)) \leq \mathrm{hn}(G) \leq 2\,\mathrm{hn}(\mathrm{IG}(G)).$$

Proof. Let $\mathrm{IG}(G)$ be the incidence graph of G. Observe that any hull set of G is a hull set of $\mathrm{IG}(G)$, since for any shortest path, $P = \{v_1, \ldots, v_k\}$ in G there is a shortest path $P' = \{v_1, s_{v_1v_2}, v_2, \ldots, s_{v_{k-1}v_k}, v_k\}$ in $\mathrm{IG}(G)$ (the edges were subdivided). Consequently, $\mathrm{hn}(\mathrm{IG}(G)) \leq \mathrm{hn}(G)$. However, given a hull set S_h of $\mathrm{IG}(G)$, one may find a hull set of G by simply replacing each vertex of S_h that represents an edge of G by its neighbors (vertices of G). Thus, $\mathrm{hn}(G) \leq 2\,\mathrm{hn}(\mathrm{IG}(G))$. □

Corollary 5. *If there exists a k-approximation algorithm B to compute the hull number of bipartite graphs, then B is a $2k$-approximation algorithm for any graph.*

9.3 Complement of Bipartite Graphs

A graph $G = (V, E)$ is a *complement of a bipartite graph* if there is a partition $V = A \cup B$ such that A and B are cliques. In this section, we give a polynomial-time algorithm to compute a hull set of G with size $\mathrm{hn}(G)$. We start with some notation.

Given the partition (A, B) of V, we say that an edge $\{u, v\} \in E$ is a *crossing-edge* if $u \in A$ and $v \in B$. Denote by S the set of simplicial vertices of G. Let $S_A = S \cap A$ and by $S_B = S \cap B$. Let U be the set of universal vertices of G. Note that, if G is not a clique, $U \cap S = \emptyset$. Let H be the graph obtained from G by removing the vertices in S and U, and removing the edges intra-clique, i.e., $V(H) = V \setminus (U \cup S)$ and $E(H) = \{\{u, v\} \in E \mid u \in A \cap V(H) \text{ and } v \in B \cap V(H)\}$. Let $\mathcal{C} = \{C_1, \ldots, C_r\}$ $(r \geq 1)$ denote the set of connected components C_i of H. Observe that, if G is neither one clique nor the disjoint union of A and B, H is not empty and each connected component C_i has at least two vertices, for every $i \in \{1, \ldots, r\}$. Indeed, any vertex in $A \setminus S_A$ (resp., in $B \setminus S_B$) has a neighbor in $B \cap V(H)$ (resp. in $A \cap V(H)$).

Theorem 71. *Let $G = (A \cup B, E)$ be the complement of a bipartite n-node graph. There is an algorithm that computes $\mathrm{hn}(G)$ and a hull set of this size in time $O(n^7)$.*

Proof. We use the notations defined above. Recall that, by Lemma 63, S is contained in any hull set of G. In particular, if G is a clique or G is the disjoint union of two cliques A and B, then $\mathrm{hn}(G) = n$. From now on, we assume it is not the case. By Lemma 64, no vertices in U belong to any minimal hull set of G. Now, several cases have to be considered.

Claim 21. *If $U = \emptyset$, $S_A \neq \emptyset$ and $S_B \neq \emptyset$, then S is a minimum hull set of G and thus $\mathrm{hn}(G) = |S|$.*

Proof. Since G has no universal vertex, a simplicial vertex in S_A (in S_B) has no neighbor in B (resp., in A). Since G is not the disjoint union of two cliques, every vertex $u \in A \setminus S_A$ has a neighbor $v \in B \setminus S_B$ and vice-versa. Thus, $\{s_a, u, v, s_b\}$ is a shortest (s_a, s_b)-path, for any $s_a \in A$ and $s_b \in B$, and then $u, v \in I_h[S]$. □

Hence, from now on, let us assume that $U \neq \emptyset$ or, w.l.o.g., $S_B = \emptyset$.

Again, if there is some simplicial vertex in G, i.e., if $S_A \neq \emptyset$, all the vertices of S belong to any hull set of G and thus $\mathrm{hn}(G) \geq |S|$. In fact, for each connected component of H, we prove that it is necessary to choose at least one of its vertices to be part of any hull set of G.

Claim 22. *If $U \neq \emptyset$ or $S_B = \emptyset$ or $S_A = \emptyset$, then $\mathrm{hn}(G) \geq |S| + r$.*

Proof. Again, all vertices of S belong to any hull set of G. We show that, for any $1 \leq i \leq r$, $V \setminus C_i$ is a convex set. Thus, by Lemma 66, any hull set of G contains at least one vertex of C_i for any $i \leq r$.

It is sufficient to show that no pair $u, v \in V(G) \setminus C_i$ can generate a vertex v_i of C_i. By contradiction, suppose that there exists a pair of vertices $u, v \in V(G) \setminus C_i$ such that there is a shortest (u, v)-path P containing a vertex v_i of C_i. Consequently, u and v must not be adjacent and we consider that $u \in A$ and $v \in B$. If $U = \emptyset$, then, w.l.o.g., $S_B = \emptyset$

and v is not simplicial and has at least one neighbor in A. Hence, since $U \neq \emptyset$ or $S_b = \emptyset$, u and v are at distance two. Consequently, $P = \{u, v_i, v\}$. However, if $v_i \in A$, v belongs to C_i, because of the crossing edge $\{v_i, v\}$, otherwise, $u \in C_i$. In both cases we reach a contradiction. □

Now, two cases remain to be considered. We recall that $U \neq \emptyset$ or $S_B = \emptyset$.

1. If $r \geq 2$, then $\mathrm{hn}(G) = |S| + r$, and we can build a minimum convex hull by taking the vertices in S, one arbitrary vertex in $A \cap C_i$ for all $i < r$ and one arbitrary vertex in $B \cap C_r$.

   Let $R = \{v_1, \ldots, v_r\}$ such that $v_i \in C_i \cap A$ for any $i < r$ and $v_r \in C_r \cap B$.

   Claim 23. *$S \cup R$ is a hull set of G.*

   Proof. Since all vertices in U are generated by v_1 and v_r (that are not adjacent, since they are in different components), it is sufficient to show that $S \cup R$ generates all the vertices in C_i, for any $i \in \{1, \ldots, r\}$. Actually, we show that R generates all the vertices in C_i.

   By contradiction, suppose that there is a vertex $z \notin I_h[R]$. Let $i \leq r$ such that $z \in C_i$. Because C_i contains one vertex in R and is connected, we can choose z and $w \in C_i \cap I_h[R]$ linked by a crossing edge. We will show that $z \in I_h[R]$ (a contradiction), hence, w.l.o.g., we may assume that $z \in A$. If $i = r$, then $\{v_1, z, w\}$ is a shortest (v_1, w)-path and $z \in I_h[R]$.

   Otherwise, recall that $N(v_r) \cap A \cap C_r \neq \emptyset$ and, for any $i < r$, $N(v_i) \cap B \cap C_i \neq \emptyset$ because v_i is not simplicial for any $i \leq r$. Let $x \in N(v_r) \cap A \cap C_r$ and $y_i \in N(v_i) \cap B \cap C_i$. Note that $x \in I_h[R]$ because $\{v_1, x, v_r\}$ is a shortest (v_r, v_1)-path, and $y_i \in I_h[R]$ because $\{v_i, y_i, v_r\}$ is a shortest (v_r, v_i)-path. Hence, since $\{x, z, y_i\}$ is a shortest (x, y_i)-path, we have $z \in I_h[R]$. □

   As $|R| = r$, we conclude by Claim 22 that $\mathrm{hn}(G) = |S| + r$.

2. If $r = 1$, then $\mathrm{hn}(G) \leq |S| + 4$, and any minimum convex hull contains at most 4 vertices not in S.

   Again, S is included in any hull set of G by Lemma 63, and no vertices in U belong to some hull set by Lemma 64. In this case, when H has just one connected component $C_1 = C$, one vertex of C may not suffice to generate this component, as in the previous case. However, we prove that at most 4 vertices in C are needed.

   a) If $S_A \neq \emptyset$ and $S_B \neq \emptyset$ (and thus $U \neq \emptyset$ because Claim 21 applies otherwise), then $\mathrm{hn}(G) = |S| + 1$.

      By Claim 22, we know that $\mathrm{hn}(G) \geq |S| + 1$. Let v be an arbitrary vertex of C. We claim that $S \cup \{v\}$ is a minimum hull set of G. By contradiction, let $z \notin I_h[S \cup \{v\}]$. Since C is a connected component of H, we may choose z such that there is $w \in N(z) \cap C \cap I_h[S \cup \{v\}]$. Moreover, we may assume w.l.o.g. that $z \in A$, and thus $w \in B$. In that case, since $S_A \neq \emptyset$, there is $v_A \in S_A$ and as $\{v_A, w\} \notin E(G)$ (indeed, any vertex in $N(v_A) \cap B$ must be universal because v_A is simplicial, which is not the case since w is not universal because it belongs to C), z is generated by v_A and w.

b) If $S_A \neq \emptyset$ and $S_B = \emptyset$, then $\text{hn}(G) \leq |S| + 2$. Let $v_A \in A \cap C$ be such that $|N(v_A) \cap B \cap C|$ is maximum. Since v_A is not universal in G, there exists $x \in B$ such that $\{v_A, x\} \notin E(G)$. Note that $x \in C$ since x is not universal and $S_B = \emptyset$. Let $R = \{v_A, x\}$. Observe that $N(v_A) \cap B \cap C \subseteq I_h[R \cup S]$ since $\{v_A, x\} \notin E$.

By contradiction, assume $V(G) \setminus I_h[R \cup S] \neq \emptyset$. Let $z \in V(G) \setminus I_h[R \cup S]$. First, suppose that $z \in A$. Since C is connected in H, we may assume that z has a neighbor $w \in I_h[R \cup S] \cap B \cap C$. As $S_A \neq \emptyset$, there is $v \in S_A$ and as $\{v, w\} \notin E(G)$ (because otherwise w would be universal in G and not in C), z is generated by v and w. Now suppose that $z \in B$, and now it has a neighbor $w \in I_h[R \cup S] \cap A \cap C$. Observe that $I_h[R \cup S] \cap B \subseteq N(w)$, otherwise z would be in $I_h[R \cup S]$. However, since $N(v_A) \cap B \cap C \subset (N(v_A) \cap B \cap C) \cup \{x\} \subseteq I_h[R \cup S] \cap B$, we get that $N(v_A) \cap B \cap C \subset N(w) \cap B \cap C$, contradicting the maximality of $|N(v_A) \cap B \cap C|$.

c) If $S_A = \emptyset$ and $S_B = \emptyset$, then $\text{hn}(G) \leq 4$.

Let $v_A \in A \cap C$ be such that $|N(v_A) \cap B \cap C|$ is maximum and $v_B \in B \cap C$ be such that $|N(v_B) \cap A \cap C|$ is maximum. Since v_A is not universal in G and $S_B = \emptyset$, there exists $y \in C \cap B \setminus N(v_a)$, and similarly there exists $x \in C \cap A \setminus N(v_B)$. Let $R = \{v_A, v_B, x, y\}$. Observe that $N(v_A) \cap B \subseteq I_h[R]$ and $N(v_B) \cap A \subseteq I_h[R]$, since $\{v_A, y\} \notin E$ and $v_B x \notin E$.

By contradiction, assume $V(G) \setminus I_h[R] \neq \emptyset$. Let $z \in V(G) \setminus I_h[R]$. First, suppose that $z \in A$. As in the previous case, since C is connected in H, we may assume that z has a neighbor $w \in I_h[R] \cap B \cap C$. Observe that $I_h[R] \cap A \cap C \subseteq N(w)$, otherwise z would be in $I_h[R]$. However, since $N(v_B) \cap A \cap C \subset (N(v_B) \cap A \cap C) \cup \{x\} \subseteq I_h[R] \cap A \cap C$, we get that $N(v_B) \cap A \cap C \subset N(w) \cap A \cap C$, contradicting the maximality of $|N(v_B) \cap A \cap C|$.

Whenever $z \in B$, one can use the same arguments to reach a contradiction on the maximality of $|N(v_A) \cap B \cap C|$.

Since $|S| + 1 \leq \text{hn}(G) \leq |S| + 4$, S is included in any hull set of G and no vertices in U belong to some hull set, there exist a subset R of at most 4 vertices in C such that $S \cup R$ is a minimum hull set of G. There are $O(|V|^4)$ subsets to be tested and, for each one, its convex hull can be computed in $O(|V||E|)$ time [DGK$^+$09]. This leads to the announced result. □

9.4 Graphs with few P_4's

As stated in Chapter 8, the hull number of a cograph can be computed in polynomial time [DGK$^+$09]. Since cographs are graphs that have no P_4 as induced subgraph, in this section we investigate the complexity of computing the hull number of a graph that has "few" induced P_4.

A graph $G = (V, E)$ is a $(q, q-4)$-graph, for a fixed $q \geq 4$, if for any $S \subseteq V$, $|S| \leq q$, then S induces at most $q-4$ paths on 4 vertices. Observe that cographs are the $(4, 0)$-graphs.

In this section, we generalize these results by proving that for any fixed $q \geq 4$, computing the hull number of a $(q, q-4)$-graph can be done in polynomial time. Our algorithm runs in time $O(2^q n^2)$ and is therefore a Fixed Parameter Tractable for any graph G, where the number of induced P_4's of G is the parameter.

Definitions and brief description of the algorithm

The algorithm that we present in this section uses the canonical decomposition of $(q, q-4)$-graphs, called *Primeval Decomposition*. For a survey on Primeval Decomposition, the reader is referred to [BO99]. In order to present this decomposition of $(q, q-4)$-graphs, we need the following definitions.

Let G_1 and G_2 be two graphs. $G_1 \cup G_2$ denotes the disjoint union of G_1 and G_2. $G_1 \oplus G_2$ denotes the join of G_1 and G_2, i.e., the graph obtained from $G_1 \cup G_2$ by adding an edge between any two vertices $v \in V(G_1)$ and $w \in V(G_2)$. A *spider* $G = (S, K, R, E)$ is a graph with vertex set $V = S \cup K \cup R$ and edge set E such that:

1. (S, K, R) is a partition of V and R may be empty;
2. the subgraph $G[K \cup R]$ induced by K and R is the join $K \oplus R$, and K separates S and R, i.e., any path from a vertex in S to a vertex in R contains a vertex in K;
3. S is a stable set, K is a clique, $|S| = |K| \geq 2$, and there exists a bijection $f : S \to K$ such that, either $N(s) \cap K = K - \{f(s)\}$ for all vertices $s \in S$, or $N(s) \cap K = \{f(s)\}$ for all vertices $s \in S$. In the latter case or if $|S| = |K| = 2$, G is called *thin*, otherwise G is *thick*.

A graph $G = (S, K, R, E)$ is a *pseudo-spider* if it satisfies only the first two properties of a spider. A graph $G = (S, K, R, E)$ is a *q-pseudo-spider* if it is a pseudo-spider and, moreover, $|S \cup K| \leq q$. Note that q-pseudo-spiders and spiders are pseudo-spiders.

We now describe the decomposition of $(q, q-4)$-graphs.

Theorem 72 ([BO99])**.** *Let $q \geq 0$ and let G be a $(q, q-4)$-graph. Then, one of the following holds:*

1. *G is a single vertex, or*
2. *$G = G_1 \cup G_2$ is the disjoint union of two $(q, q-4)$-graphs G_1 and G_2, or*
3. *$G = G_1 \oplus G_2$ is the join of two $(q, q-4)$-graphs G_1 and G_2, or*
4. *G is a spider (S, K, R, E) where $G[R]$ is a $(q, q-4)$-graph if $R \neq \emptyset$, or*
5. *G is a q-pseudo-spider (H_2, H_1, R, E) where $G[R]$ is a $(q, q-4)$-graph if $R \neq \emptyset$.*

Theorem 72 leads to a tree-like structure $T(G)$ (the *primeval tree*) which represents the Primeval Decomposition of a $(q, q-4)$-graph G. $T(G)$ is a rooted binary tree where any vertex v corresponds to an induced $(q, q-4)$-subgraph G_v of G and the root corresponds to G itself. Moreover, the vertices of subgraphs corresponding to the leaves of $T(G)$ form a partition of $V(G)$, i.e., $\{V(G_\ell)\}_{\ell \text{ leaf of } T(G)}$ is a partition of $V(G)$.

For any leaf ℓ of $T(G)$, G_ℓ is either a spider $(S, K, \emptyset, E)$, or has at most q vertices. Moreover, any internal vertex v has its label following one of the four cases in Theorem 72 corresponds to G_v. More precisely, let v be an internal vertex of $T(G)$ and let u and w be its two children. v is a *parallel node* if $G_v = G_u \cup G_w$. v is a *series node* if $G_v = G_u \oplus G_w$. v is a *spider node* if u is a leaf with G_u is a spider $(S, K, \emptyset, F)$ and G_v is the spider (S, K, R, E) where $G_v[R] = G_w$ and $G_v[S \cup K] = G_u$. Finally, v is a *small node* if u is a leaf with $|V(G_u)| \leq q$ and G_v is the q-pseudo-spider (S, K, R, E) where $G_v[R] = G_w$ and $G_v[S \cup K] = G_u$.

This tree can be obtained in linear-time [BO99].

We compute $\mathrm{hn}(G)$ by a post-order traversal in $T(G)$. More precisely, given $v \in V(T(G))$, let H_v be an optimal hull set of G_v and let H_v^* be an optimal hull set of G_v^*, the graph obtained by adding a universal vertex to G_v. We show in the next subsection that we can compute (H_ℓ, H_ℓ^*) for any leaf ℓ of $T(G)$ in time $O(2^q n)$. Moreover, for any internal vertex v of $T(G)$, we show that we can compute (H_v, H_v^*) in time $O(2^q n)$, using the information that was computed for the children and grand children of v in $T(G)$.

Theorem 73. *Let $q \geq 0$ and let G be a n-node $(q, q-4)$-graph. An optimal hull set of G can be computed in time $O(2^q n^2)$.*

Before going into the details of the algorithm in next subsection, we prove some useful lemmas.

Lemma 74. *Let $G = (S, K, R, E)$ be a pseudo-spider with R neither empty nor a clique. Then any minimum hull set of G contains a minimum hull set of the subgraph $G[K \cup R]$.*

Proof. Let H be a minimum hull set of G. Let $H_S = H \cap S$ and $H_R = H \setminus H_S$. We prove that H_R is a minimum hull set of $G[K \cup R]$.

Let H' be any minimum hull set of $G[K \cup R]$. Note that $H' \subseteq R$ because K is a set of universal vertices in $G[K \cup R]$ and by Lemma 64. Moreover, By Lemma 65, because $G[K \cup R]$ is an isometric subgraph of G, the convex hull of H' in G contains $G[K \cup R]$. Hence, $H_S \cup H'$ is a hull set of G and $\mathrm{hn}(G) \leq |H_S| + \mathrm{hn}(G[K \cup R])$.

Now it remains to prove that H_R is a hull set of $G[K \cup R]$. Clearly, if H_R generate all vertices of R in $G[K \cup R]$ then H_R is a hull set of $G[K \cup R]$ since there are at least two non adjacent vertices in R and any vertex in K is adjacent to all vertices in R. For purpose of contradiction, assume H_R does not generate R in $G[K \cup R]$. This means that there is a vertex $v \in R$, that is generated in G by a vertex in $S \cup K$, i.e., $v \in R$ is an internal vertex of a shortest path between $s \in S \cup K$ and some other vertex, which is not possible, since we have all the edges between K and R. Hence, $\mathrm{hn}(G[K \cup R]) \leq |H_R|$.

Therefore, $|H_S| + |H_R| = \mathrm{hn}(G) \leq |H_S| + \mathrm{hn}(G[K \cup R]) \leq |H_S| + |H_R|$. So, $\mathrm{hn}(G[K \cup R]) = |H_R|$, i.e., H_R is a minimum hull set of $G[K \cup R]$ contained in H. □

The next lemma is straightforward by the use of isometry.

Lemma 75. *Let G be a graph which is not complete and that has a universal vertex. Let H be obtained from G by adding some new universal vertices. A set is a minimum hull set of G if, and only if, it is a minimum hull set of H.*

Dynamic programming and correctness

In this section, we detail the algorithm presented in the previous section and we prove its correctness. Let $v \in V(T(G))$, which may therefore be either a leaf, a parallel node, a series node, a spider node or a small node. For each of these five cases, we describe how to compute (H_v, H_v^*), in time $O(2^q n)$.

Let us first consider the case when v is a leaf of $T(G)$.

If G_v is a singleton $\{w\}$, then $H_v = V(G_v) = \{w\}$ and $H_v^* = V(G_v^*)$. If G_v is a spider $(S, K, \emptyset, E)$ then $H_v = S$ since S is a set of simplicial vertices (so it has to be included in any hull set by Lemma 63) and it is sufficient to generate G_v. One may easily check that if G_v is a thick spider, S is also a minimum hull set of G_v^*, i.e., $S = H_v^*$. However, in case G_v is a thin spider, S does not suffice to generate G_v^* and in this case it is easy to see that this is done by taking any extra vertex $k \in K$, in which case we have $H_v^* = S \cup \{k\}$.

Finally, if G_v has at most q vertices, H_v and H_v^* can be computed in time $O(2^q)$ by an exhaustive search.

Now, let v be an internal node of $T(G)$ with children u and w.

If v is a parallel node, then $G_v = G_u \cup G_w$. Then, (H_v, H_v^*) can be computed in time $O(1)$ from (H_u, H_u^*) and (H_w, H_w^*) thanks to Lemma 76.

Lemma 76 ([DGK+09])**.** *Let $G_v = G_u \cup G_w$. Then $(H_v, H_v^*) = (H_u \cup H_w, H_u^* \cup H_w^*)$.*

Proof. The fact that $H_u \cup H_w$ is an optimal hull set for G_v is trivial. The second part comes from the fact that H_u^* (resp., H_w^*) is an isometric subgraph of H_v^* and from Lemma 65. □

Now, we consider the case when v is a series node.

Lemma 77. *If $G_v = G_u \oplus G_w$, then (H_v, H_v^*) can be computed from the sets (H_x, H_x^*) of the children or grand children x of v in $T(G)$, in time $O(2^q n)$.*

Proof. If G_u and G_w are both complete, then the graph G_v is a clique and $(H_v, H_v^*) = (V(G_v), V(G_v^*))$.

If G_u and G_w are both not complete, let x, y be any two non adjacent vertices in G_u. Then, we claim that $H_v = H_v^* = \{x, y\}$. Indeed, in G_v, x and y generate all vertices in $V(G_w)$ (resp., of G_w^*). In particular, two non adjacent vertices $z, r \in V(G_w)$ are generated. Symmetrically, z, r generate all vertices in $V(G_u)$ (resp., in $V(G_u^*)$).

Without loss of generality, we suppose now that G_u is a complete graph and that G_w is a non-complete $(q, q-4)$-graph. First, observe that no vertex of G_u belongs to any minimum hull set of G_v, since they are universal (Lemma 64). Note also that, by Lemma 75 and since G_v is not a clique and has universal vertices, we can make $H_v = H_v^*$. Hence, in what follows, we consider only the computation of H_v. Let us consider all possible cases for w in $T(G)$.

- w is a series node. G_w is the join of two graphs. We claim that $H_v = H_w$.

  In this case, the graph G_w is an isometric subgraph of G_v. Thus, by Lemma 65, any minimum hull set of G_w generates all vertices of $V(G_w)$ in G_v. Finally, since G_w has two non-adjacent vertices they generate all vertices of G_u in G_v.

- w is a parallel node. G_w is the disjoint union of two graphs. Let x and y be the children of w in $T(G)$. Then $G_w = G_x \cup G_y$. Let $X = H_x^*$ if G_x is not a clique and $X = V(G_x)$, otherwise, let $Y = H_y^*$ if G_y is not a clique and $Y = V(G_y)$, otherwise. We claim that $H_v = X \cup Y$.

  Clearly, if G_x (resp., G_y) is a clique, all its vertices are simplicial in G_v and then must be contained in any hull set by Lemma 63. Moreover, recall that, by Lemma 64, no vertex of G_u belongs to any minimum hull set of G.

  Now, let $z \in \{x, y\}$ such that G_z is not complete. It remains to show that it is necessary and sufficient to also include any minimum hull set H_z^* of G_z^*, in any minimum hull set of G.

  The necessity can be easily proved by using Lemma 74 to every G_z that is not a complete graph.

  The sufficiency follows again from the fact that G_u is generated by two non adjacent vertices of G_w and since, in all cases, $X \cup Y$ contains at least one vertex in G_x and one vertex in G_y, all vertices in G_u will be generated.

- w is a spider node and G_w is a thin spider $(S, K, \emptyset, E')$. Then, $H_v = S \cup \{k\} = G^*_w$ where k is any vertex in K.

  All vertices in S are simplicial in G_v, hence any hull set of G_v must contain S by Lemma 63. Now, in G_v, the vertices in S are at distance two and no shortest path between two vertices in S passes through a vertex in K, since there is a join to a complete graph. Therefore, S is not a hull set of G_v. However, since $|S| \geq 2$, it is easy to check that adding any vertex $k \in K$ to S is sufficient to generate all vertices in G_v. So $S \cup \{k\}$ is a minimum hull set of G_v.

  Note that, in that way, $H_v = S \cup \{k\} = G^*_w$.

- w is a spider node and G_w is a spider (S, K, R, E') that is either thick or $R \neq \emptyset$ and R induces a $(q, q-4)$-graph. Then, $H_v = H_w$.

  If $R = \emptyset$, then G_w is thick. In this case, it is easy to check that the only minimum hull set of G_w is S (because it consists of simplicial vertices) and it is also a minimum hull set for G_v. Hence, $H_v = H_w = S$.

  If $R \neq \emptyset$, then by Lemma 63 any minimum hull set of G_w contains S. Moreover, by Lemma 74 any minimum hull set of G_w contains a minimum hull set of $K \cup R$ which is composed by vertices of R.

  By the same lemmas, a minimum hull set of G_w is a minimum hull set of G_v since, by Lemma 64, no vertex of G_u belongs to any minimum hull set of G_v and G_u is generated by non-adjacent vertices of G_w.

- w is a small node. G_w is a q-pseudo-spider (H_2, H_1, R, E') and R induces a $(q, q-4)$-graph.

  If $R = \emptyset$, G_v is the join of a clique G_u with a graph G_w that has at most q vertices. No vertex of G_u belongs to any minimum hull set of G_v, since they are universal. Thus, H_v can be computed in time $O(2^q)$ by testing all the possible subsets of vertices of G_w.

  Similarly, if R is a clique, all vertices in R are simplicial in G_v so they must belong to any hull set of G_v. Moreover, no vertices in G_u belong to any minimum hull set of G_v. So H_v can be computed in time $O(2^q)$ by testing all the possible subsets of vertices of $H_1 \cup H_2$ and adding R to them.

  In case $R \neq \emptyset$ nor a clique, two cases must be considered. By definition of the decomposition, there exists a child r of w in $T(G)$ such that $V(G_r) = R$.

  - If $G[H_1]$ is a clique, then, $G_v = (H_2, H_1 \cup V(G_u), R, E)$ is a pseudo-spider that satisfies the conditions in Lemma 74. Hence, any minimum hull set of G_v contains a minimum hull set of $P = G[H_1 \cup V(G_u) \cup R]$. Let Z be a minimum hull set of G_v and let $Z' = Z \cap H_2$. By Lemma 74, we have $|Z'| \leq \mathrm{hn}(G_v) - \mathrm{hn}(P)$.

    By Lemma 75, H^*_r is a minimum hull set of $G[H_1 \cup V(G_u) \cup R]$. Now, $G[H_1 \cup V(G_u) \cup R]$ is an isometric subgraph of G_v. Hence, by Lemma 65, H^*_r generates all vertices of $G[H_1 \cup V(G_u) \cup R]$ in G_v. Therefore, $H^*_r \cup Z'$ will generate all vertices of G_v. Since $|H^*_r| = \mathrm{hn}(P)$, we get that $|H^*_r \cup Z'| \leq \mathrm{hn}(G_v)$ and then $H^*_r \cup Z'$ is a minimum hull set of G_v.

So, we have shown that there exists a minimum hull set for G_v that can be obtained from H_r^* by adding some vertices in $H_1 \cup H_2$. Since $|H_1 \cup H_2| \leq q$, such a subset of $H_1 \cup H_2$ can be found in time $O(2^q)$.

– In case $G[H_1]$ is not a clique, let x and y be two non adjacent vertices of H_1. We claim in this case that there exists a minimum hull set of G_v containing at most one vertex of R. Let S be a minimum hull set of G_v containing at least two vertices in R. Observe that $S' = (S \setminus R) \cup \{x, y\}$ is also a hull set of G_v since x and y are sufficient to generate all vertices in R. Consequently, $|S'| \leq |S|$ and S' is minimum.

  Since no hull set of G_v contains a vertex in $V(G_u)$, there always exists a minimum hull set of G_v that consists of only vertices in $H_1 \cup H_2$ plus at most one vertex in R. Therefore an exhaustive search can be performed in time $O(2^q n)$.

□

Now, we consider the case when v is a spider node or a small node. That is $G_v = (S, K, R, E)$. If $R \neq \emptyset$, let r be the child of v such that $V(G_r) = R$.

Lemma 78. *Let $G_v = (S, K, R, E)$ be a spider such that R induces a $(q, q-4)$-graph.*

Then, $H_v = H_v^* = S \cup H_r^*$ if $R \neq \emptyset$ and R is not a clique, and $H_v = H_v^* = S \cup R$, otherwise.

Proof. Since all the vertices in S are simplicial vertices in G_v and in G_v^*, we apply Lemma 63 to conclude that they are all contained in any hull set of G_v (resp., of G_v^*).

By the structure of a spider, every vertex of K (and the universal vertex in G_v^*) belongs to a shortest path between two vertices in S and are therefore generated by them in any minimum hull set of G_v (resp., of G_v^*). Consequently, if $R = \emptyset$, S is a minimum hull set of G_v (resp., of G_v^*). If R is a clique, $S \cup R$ is the set of simplicial vertices of G_v (resp., of G_v^*) and also a minimum hull set of G_v (resp., of G_v^*).

Finally, if $R \neq \emptyset$ and R is not a clique, then G_v is a pseudo-spider satisfying the conditions of Lemma 74. Similarly, G_v^* is a pseudo-spider (by including the universal vertex in K). Then, by Lemma 74, any hull set of G_v (resp., of G_v^*) contains a minimum hull set of $G[K \cup R]$ (resp., of $G_v^* \setminus S$. Moreover, any hull set contains all vertices in S since they are simplicial. Hence, $\mathrm{hn}(G_v) = \mathrm{hn}(G_v^*) = |S| + \mathrm{hn}(G[K \cup R])$ (recall that, by Lemma 75, $\mathrm{hn}(G[K \cup R]) = \mathrm{hn}(G_v^* \setminus S)$). Finally, since $G[K \cup R])$ is an isometric subgraph of G_v, then H_r^* (which is a minimum hull set of $G[K \cup R]$ by Lemma 75) generates $G[K \cup R]$ in G_v (resp., in G_v^*).

Hence, $S \cup H_r^*$ is a hull set of G_v and G_v^*. Moreover, it has size $|S| + \mathrm{hn}(G[K \cup R])$, so it is optimal. □

Lemma 79. *Let $G_v = (H_2, H_1, R, E)$ be a q-pseudo-spider such that R is a $(q, q-4)$-graph.*

Then, H_v and H_v^* can be computed in time $O(2^q n)$.

Proof. All the arguments to prove this lemma are in the proof of Lemma 77. Moreover, the following arguments hold both for G_v and G_v^*: they allow computation of both H_v and H_v^*.

If $R = \emptyset$, G_v has at most q vertices, for a fixed positive integer q. Thus, its hull number can be computed in $O(2^q)$-time.

Otherwise, if H_1 is a clique, by Lemma 74, any minimum hull set of G_v contains a minimum hull set of $G[H_1 \cup R]$. Moreover, by the same arguments as in Lemma 77, we can show that there is an optimal hull set for G_v that can be obtained from H_r^* (minimum hull set of $G[H_1 \cup R]$) and some vertices in H_2.

If H_1 is not a clique, two non-adjacent vertices of H_1 can generate R. Thus, we conclude that there exists a minimum hull set of G_v containing at most one vertex of R. Then, a minimum hull set of G_v can be found in $O(2^q n)$-time, where $n = |V(G_v)|$. □

9.5 $\{P_5, K_3\}$-Free Graphs

In this section, we present a linear-time algorithm to compute $\mathrm{hn}(G)$, for any P_5-free triangle-free graph G.

Recall that a *dominating set* $S \subseteq V$ of a graph G is such that every vertex $v \in V \setminus S$ has a neighbor in S. Theorem 80 is one of the main building blocks in order to prove the correctness of this algorithm.

Theorem 80 ([BT90])**.** *G is P_5-free if, and only if, for every induced subgraph $H \subseteq G$ either $V(H)$ contains a dominating induced C_5 or a dominating clique.*

As a consequence, we have that:

Corollary 6. *If G is a connected P_5-free bipartite graph, then there exists a dominating edge in G.*

Theorem 81. *The hull number of a P_5-free bipartite graph $G = (A \cup B, E)$ can be computed in linear time.*

Proof. Assume that G has at least two vertices. By Corollary 6, G has at least one dominating edge. Observe that the dominating edges of a bipartite graph can be found in linear time by computing the degree of each vertex and then considering the sum of the degrees of the endpoints of each edge. For a dominating edge, this sum is equal to the number of vertices.

- Consider first the case in which G has at least two dominating edges. Let $\{u, v\}, \{x, y\} \in E(G)$ be such dominating edges. Consider that $u, x \in A$ and $v, y \in B$.

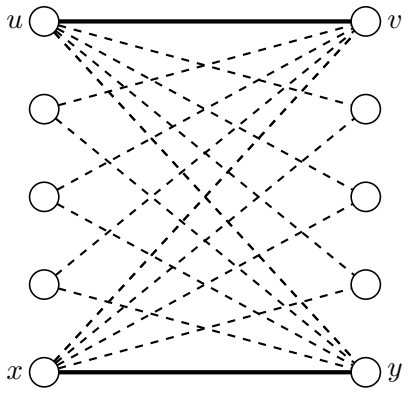


Figure 9.2: Example of two dominating edges on a bipartite graph. Dominating edges are represented by thick edges.

  If $x \neq u$ and $v \neq y$, then we claim that $\{u, x\}$ is a minimum hull set of G. An example of this case can be found in Figure 9.2. Indeed, since u and x are not

adjacent and every vertex in B is a common neighbor of u and x, and then $\{u, x\}$ generate all the vertices in B, particularly v and y. Similarly, all the vertices of A are in a shortest (v, y)-path. Thus, $I_h(\{u, x\}) = V(G)$. Therefore, $\{u, x\}$ is a hull set for G and $\mathrm{hn}(G) = 2$.

Assume now, w.l.o.g., that $u \neq x$ and $v = y$. An example of this case can be found in Figure 9.3.

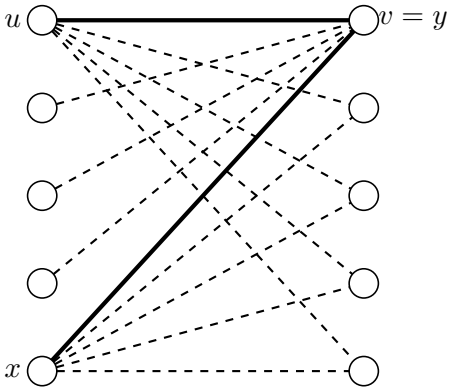


Figure 9.3: Example of two dominating edges on a bipartite graph that share one same endpoint. Dominating edges are represented by thick edges.

Again, $B \subseteq I_h(\{u, x\})$. Observe that, if there are simplicial vertices[1] in $V(G)$, they must all belong to A, since u and x are not neighbors, but they are adjacent to all vertices in B. In case $|B| = 1$, then all vertices in A are simplicial vertices, and therefore A is the minimum hull set of G.

Then, consider now that $|B| \geq 2$.

In case there is no simplicial vertex in A, $\{u, x\}$ is a minimum hull set, since $B \subseteq I_h(\{u, x\})$ and every vertex in A has at least two neighbors in B. In case there are simplicial vertices in A, we claim that $S \cup \{b\}$ is a minimum hull set of G, where $S \subset A$ is the set of simplicial vertices of G and b is a vertex in B distinct from v. Indeed, by Lemma 63, we know that S must be part of any hull set of G and observe that $I_h(S) = S \cup \{v\}$ (the only neighbor of each simplicial vertex is exactly v). Consequently, since $|B| \geq 2$, at least one more vertex must be chosen to be part of a minimum hull set of G. We claim that if we choose any arbitrary $b \in B \setminus \{v\}$, then $S \cup \{b\}$ is a minimum hull set of G. Indeed, let $s \in S$. Since $\{s, b\} \notin E$ and $\{x, v\}$, $\{u, v\}$ are dominating edges, x, u and v are generated by $\{s, b\}$. But then, as $B \subseteq I_h(\{u, x\})$, B is generated. Finally, every vertex in A is either simplicial, in case it belongs to S, or is adjacent to two vertices in B and therefore is generated by its neighbors.

- Consider now that G has only one dominating edge $\{u, v\}$ and that, w.l.o.g., $u \in A$ and $v \in B$. Let $H = G[V \setminus \{u, v\}]$. An example of this case can be found in Figure 9.4.

  The proof of this case uses the same techniques of Theorem 71. That is, we decompose G based on the connected components of some particular subgraph of G.

[1] Since G is bipartite, these vertices have degree one.

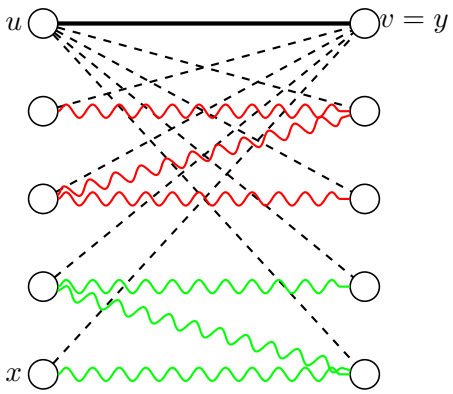


Figure 9.4: Example of a bipartite graph with only one dominating edge. The dominating edge is represented by a thick edges. Two connected component of H are represented by colored edges.

We may assume H is not the empty graph, for otherwise G is just one edge. Let $C_1, \ldots, C_k$, $k \geq 1$, be the connected components of H. We claim that $V \setminus C_i$ is a convex set of G, for every $i \in \{1, \ldots, k\}$.

Since C_i is a connected component in H, the only vertices in $V \setminus C_i$ that may be adjacent to a vertex in C_i are u and v. Suppose a shortest (s,t)-path P such that $s, t \in V \setminus C_i$ and containing at least one vertex of C_i. It would pass through u and v. But there is an edge between u and v, so there is a contradiction because P would not be a shortest path. Therefore, $V \setminus V(C_i)$ is convex.

Consequently, by Lemma 66, for each connected component C_i of H at least one vertex of C_i must be chosen to be part of a minimum hull set of G (observe that simplicial vertices are the particular case in which $|C_i| = 1$).

If $k = 1$, observe that G is not a complete bipartite graph, as we are assuming there is exactly one dominating edge. Let $w \in A$ and $z \in B$ be two non-adjacent vertices of $C_1 = H$. In this case, we claim that $\{w, z\}$ is a minimum hull set of G. By contradiction, suppose that there exists a vertex $p \notin I_h(\{w, z\})$. First observe that u and v belong to $I_h(\{w, z\})$. Then, w.l.o.g., we may assume that p has a neighbor q in $I_h(\{w, z\})$ which is not in $\{u, v\}$, since C_1 is a connected component in H. However, since $\{u, v\}$ is a dominating edge, either $\{q, p, u\}$ or $\{q, p, v\}$ is a shortest path between two vertices of $I_h[\{w, z\}]$ and p should belong to $I_h[\{w, z\}]$, a contradiction.

Now, suppose that $k > 1$. Let $W = \{w_1, \ldots, w_k\} \subseteq V(G)$ be such that $W \cap A \neq \emptyset$, $W \cap B \neq \emptyset$ and $w_i \in C_i$, for every $i \in \{1, \ldots, k\}$. We claim that W is a minimum hull set of G. By Lemma 66, $\mathrm{hn}(G) \geq k$, so it suffices to show that $I_h[W] = V(G)$. Observe that u and v belong to $I_h(W)$, since $W \cap A \neq \emptyset$ and $W \cap B \neq \emptyset$. Then, by contradiction, suppose that there exists a vertex $p \notin I_h[W]$ and let C_p be its connected component in H. Again, we may assume that p has a neighbor q in $I_h[\{w, z\}]$ which belongs to C_p, since C_p is a connected component and $C_p \cap W \neq \emptyset$. However, since $\{u, v\}$ is a dominating edge, either $\{q, p, u\}$ or $\{q, p, v\}$ is a shortest path in G and p should belong to $I_h[\{w, z\}]$, a contradiction. Therefore, $|W| = k = \mathrm{hn}(G)$.

Finally, observe that all these cases can be checked in linear time and thus $\mathrm{hn}(G)$ can be computed in linear time. □

For the next result, we mainly rely on the fact that the time complexity of finding the convex hull of a set of vertices $S \subseteq V(G)$ of a graph G is $O(|S||E(G)|)$, as described in [DGK+09].

Corollary 7. *If G is a P_5-free triangle-free graph, then* $\mathrm{hn}(G)$ *can be computed in* $O(|V(G)|^3|E(G)|)$.

Proof. By Theorem 80, G either has a dominating induced C_5 or a dominating clique of size at most two, since it is triangle-free.

In case it has a dominating $C_5 = v_1, \ldots, v_5$, we claim that $\{v_1, v_3, v_5\}$ is a hull set of G. To prove this fact, first observe that $I_h[\{v_1, v_3, v_5\}] \supseteq V(C_5)$. Moreover, since G is connected, and it has no induced P_5 and no triangle, we conclude that any vertex $w \in V(G) \setminus V(C_5)$ has two non-adjacent neighbors in C_5, and so $w \in I_h[\{v_1, v_3, v_5\}]$. Thus, if G has a dominating C_5, we can test if there is a minimum hull set of size two in $O(|V(G)|^2|E(G)|)$. Otherwise, we have that $\mathrm{hn}(G) = 3$ and $\{v_1, v_3, v_5\}$ is a minimum hull set of G.

If G has a dominating clique of size one, then G must be a star since it is triangle-free. Thus, $\mathrm{hn}(G) = |V(G)| - 1$.

Finally, if G has a dominating edge $\{u, v\}$, we claim that G is bipartite. Since G is triangle-free and $\{u, v\}$ is a dominating edge, we have that $N(u)$ and $N(v)$ are stable sets and that $N(u) \cap N(v) = \emptyset$. Thus, G is bipartite and, by Theorem 81, we can compute its hull number in linear time.

Considering all the cases, we have that either $\mathrm{hn}(G) \leq 3$ or $\mathrm{hn}(G) = |V(G)| - 1$. Therefore, we can test in time $O(|V(G)|^3|E(G)|)$, for each subset of $V(G)$ with at most 3 vertices, if it is a hull set for G. If there is no set $S \subseteq V(G)$ with $|S| \leq 3$ such that S is a hull set for G, then $\mathrm{hn}(G) = |V(G)| - 1$. □

9.6 Reduction Rules

In this section, we present three reduction rules to compute the hull number of a graph. We need to introduce some definitions.

Given a graph G, we say that two vertices v and v' are *twins* if $N(v)\setminus\{v'\} = N(v')\setminus\{v\}$. If v and v' are adjacent, we call them *true* twins, otherwise we say that they are *false* twins.

Let G be a graph and v and v' be two of its vertices. The *identification* of v' into v is the operation that produces a graph G' such that $V(G') = V(G) \setminus \{v'\}$ and $E(G') = (E(G) \setminus \{\{v', w\} \mid w \in N_G(v')\}) \cup \{\{v, w\} \mid \{v', w\} \in E(G) \text{ and } w \neq v\}$.

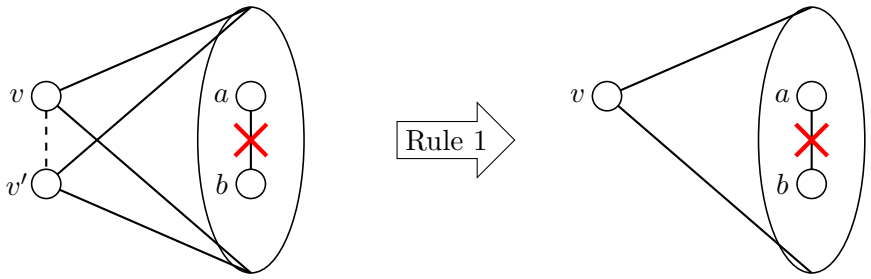


Figure 9.5: Twin vertices v and v' have their neighborhood represented by the ellipse. Vertices a and b are not adjacent. The vertex v' is identified into v after application of Rule 1.

Lemma 82 (Rule 1)**.** *Let G be a graph and v and v' be non-simplicial and twin vertices. Let G' be obtained from G by the identification of v' into v. Then,* $\mathrm{hn}(G) = \mathrm{hn}(G')$*. A scheme of this rule can be found in Figure 9.5.*

Proof. Let u and w be two non-adjacent neighbors of v and thus also of v' in G. In order to show that $\mathrm{hn}(G) \leq \mathrm{hn}(G')$, let S be a minimum hull set of G'. Since G' is an isometric subgraph of G, $V(G) \setminus \{v'\} \subseteq I_h[S]$ by Lemma 65. Moreover, $\{v'\} \subseteq I_G[u, w]$, hence S is a hull set of G.

To prove that $\mathrm{hn}(G) \geq \mathrm{hn}(G')$, let S be a minimum hull set of G. We may assume that S does not contain both v and v', because if there exists a minimum hull set containing both of them, then we can replace v and v' by u and w obtaining a hull set of same size, since $v, v' \in I_G[u, w]$.

Suppose first that $v, v' \notin S$. Let $\{x, y\} \neq \{v, v'\}$ and let P be a shortest (x, y)-path. Observe that P cannot contain both v and v'. In case v' (resp. v) is contained in P, then one can replace it by v (resp. v') and obtain another shortest path, as v and v' have the same neighborhood. In particular, this implies that the minimum k such that $v' \in I^k_G[S]$ is equal to the minimum k' such that $v \in I^{k'}_{G'}[S]$, and therefore for $i < k$, $I^i_{G'}[S] = I^i_G[S]$. It also implies that $I_G[v', w] \setminus \{v'\} = I_{G'}[v, w] \setminus \{v\}$, $w \notin \{v, v'\}$, and therefore for $i \geq k$ we have that $I^i_{G'}[S] = I^i_G[S] \setminus \{v'\}$. As a consequence, S is a hull set of G'.

Finally, suppose that either v or v' is in S. We may assume w.l.o.g. that $v \in S$. Then we can use the same argument as in the last paragraph to show that for every $1 \leq i \leq n$ its true that $I^i_{G'}[S] = I^i_G[S] \setminus \{v'\}$ and then again we have that S is a hull set of G'. □

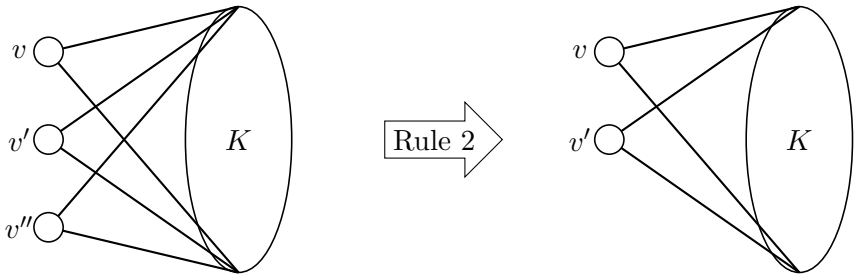


Figure 9.6: False twin vertices v, v' and v'' have their neighborhood represented by the ellipse. The vertex v'' is identified into v after application of Rule 2.

Lemma 83 (Rule 2)**.** *Let G be a graph and v, v', v'' be simplicial and pairwise false twin vertices. Let G' be obtained from G by the identification of v'' into v. Then,* $\mathrm{hn}(G) = \mathrm{hn}(G') + 1$*. A scheme of this rule can be found in Figure 9.6.*

Proof. In order to show that $\mathrm{hn}(G) \leq \mathrm{hn}(G')+1$, observe that G' is an isometric subgraph of G and that v'' is simplicial. Consequently, any hull set S of G' is such that $I_h[S] = V(G) \setminus \{v''\}$, hence $S \cup \{v''\}$ is a hull set of G, by Lemmas 63 and 65.

To show that $\mathrm{hn}(G) \geq \mathrm{hn}(G') + 1$. Let S be a hull set for G and $S' = S \setminus \{v''\}$. Since v, v' and v'' are simplicial, we know that $\{v, v', v''\} \subseteq S$. Any shortest (v'', u)-path, with $u \in V \setminus \{v', v''\}$ is still a shortest path if v'' is replaced by v', so $I[v'', u] \setminus \{v''\} = I[v', u] \setminus \{v'\}$. In the case of the shortest (v'', v')-path, replacing v'' by v is still a shortest path and $I[v'', v'] \setminus \{v''\} = I[v, v'] \setminus \{v\}$. Therefore $I_h[S'] = I_h[S] \setminus \{v''\}$ and then S' is a hull set of G'. □

Observe that we cannot simplify the statement of Lemma 83 to consider any pair of simplicial false twin vertices instead of triples. As an example, consider the graph obtained by removing an edge $\{u, v\}$ from a complete graph with more than 3 vertices.

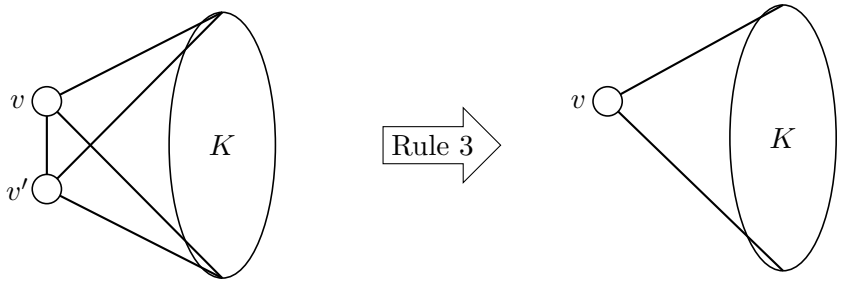


Figure 9.7: True twin vertices v and v' have their neighborhood represented by the ellipse. The vertex v' is identified into v after application of Rule 3.

Lemma 84 (Rule 3)**.** *Let G be a graph and v, v' be simplicial and true twin vertices. Let G' be obtained from G by the identification of v' into v. Then,* $\mathrm{hn}(G) = \mathrm{hn}(G') + 1$. *A scheme of this rule can be found in Figure 9.7.*

Proof. In order to show that $\mathrm{hn}(G) \leq \mathrm{hn}(G')+1$, observe that G' is an isometric subgraph of G and that v' is simplicial. Let S be a hull set of G'. Then $S \cup \{v'\}$ is a hull set of G, by Lemmas 63 and 65.

Now, we show that $\mathrm{hn}(G) \geq \mathrm{hn}(G') + 1$. Let S be a hull set of G. Since v and v' are simplicial, by Lemma 63 we know that $v, v' \in S$. Observe that, for every $w \in V(G')$, we have $I_G[v', w] \setminus \{v'\} \subseteq I_{G'}[v, w]$. Thus, $S \setminus \{v'\}$ is a hull set of G' and the result follows. □

FPT-algorithm for the Hull Number

The *neighborhood diversity* of a graph $G = (V, E)$ is k, if its vertex set can be partitioned into k sets $S_1, \ldots, S_k$, such that any pair of vertices $u, v \in S_i$ are twins. This parameter was proposed by Lampis [Lam12], motivated by the fact that a graph of bounded vertex cover also has bounded neighborhood diversity, and therefore the later parameter can be used to obtain more general results.

To see that a graph of bounded vertex cover has bounded neighborhood diversity, let G be a graph that has a vertex cover $S \subseteq V(G)$ of size k, and let $I = V(G) \setminus S$. Since S is a vertex cover, observe that I is an independent set. Therefore, vertices in I can be partitioned in at most 2^k sets (one for each possible subset of S), where each of these sets contains twin vertices, i.e. vertices having the same neighborhood in S. Moreover, the vertices in S may be partitioned in k sets of singletons, what gives a partition of the vertices of the graph into $k + 2^k$ sets of twin vertices. Then, the neighborhood diversity of the graph is at most $k + 2^k$. Many problems have been shown to be FPT when the parameter is the neighborhood diversity [Gan12].

Now, we use the concept of neighborhood diversity to obtain the following result:

Theorem 85. *Let G be a graph whose neighborhood diversity is at most k. Then, there exists an FPT-algorithm that runs in $O(4^k \,\mathrm{poly}(|V(G)|))$-time, where* $\mathrm{poly}(x) = O(x^q)$ *for some constant q, to compute* $\mathrm{hn}(G)$.

Proof. Lampis proved that a neighborhood partition of G can be constructed in time $O(\text{poly}(|V(G)|))$ [Lam12]. Observe that each part is either an independent set of false twin vertices or a clique of true twin vertices. We now use Lemmas 82, 83 and 84 to reduce each of these parts to at most two vertices.

First, in case there are parts of size greater than one consisting of non-simplicial vertices, we reduce these parts to a single vertex by the identification of its vertices. This procedure generates a graph G' whose hull number is equal to $\text{hn}(G)$, by Lemma 82.

Observe that if a vertex is simplicial, then its part is composed of simplicial vertices. In the sequence, we reduce each part of size greater than two containing only independent simplicial false twins to two vertices, by applying Lemma 83. If c identifications are done in this procedure, then the hull number of the graph G'' obtained after this procedure is $\text{hn}(G'') = \text{hn}(G') - c = \text{hn}(G) - c$.

Then, we reduce all the parts composed of pairwise adjacent simplicial true twins to one vertex, by applying Lemma 84. In the end of this procedure, we obtain a graph G''' such that $\text{hn}(G''') = \text{hn}(G'') - c' = \text{hn}(G) - c - c'$, where c' is the number of identifications that were made in this last procedure.

Observe that G''' has at most $2k$ vertices, since the neighborhood partition is of size at most k and each part is reduced to at most two vertices. Finally, we can enumerate all the subsets of $V(G''')$ (there are at most 2^{2k} of them) and test for each of these sets whether it is a hull set. Hence, we obtain $\text{hn}(G''')$ and therefore $\text{hn}(G)$.

Another consequence of this proof is to provide a kernelization algorithm and where G''' is a kernel of linear size. □

As pointed in Chapter 1, a graph of bounded vertex cover size has also bounded neighborhood diversity, therefore the previous result also holds for this parameter, but without a linear kernel.

Hull Number of Lexicographic Product of Graphs

The reduction rules can also be applied the lexicographic product of graphs. More precisely, we use Lemma 82 and Lemma 84 to characterize the hull number of lexicographic product of two graphs. Let $S(G)$ denote the set of simplicial vertices of G.

Observe that if G has a single vertex, then $\text{hn}(G \odot H) = \text{hn}(H)$. Else, we have that:

Theorem 86. *Let G be a connected graph, such that $|V(G)| \geq 2$, and let H be an arbitrary graph. Then,*

$$\text{hn}(G \odot H) = \begin{cases} 2, & \text{if } H \text{ is not complete;} \\ (|V(H)| - 1)|S(G)| + \text{hn}(G), & \text{otherwise.} \end{cases}$$

Proof. If H is not complete, since G is connected and it has at least two vertices, any two non-adjacent vertices in the same H-layer suffice to generate all the vertices of $G \odot H$.

We consider now that H is a complete graph on k vertices. First, observe that all the vertices in the same H-layer are all simplicial vertices or they are all non-simplicial vertices. Moreover, a vertex is simplicial in G if, and only if, its corresponding H-layer in $G \odot H$ is composed of simplicial vertices.

First, we obtain from $G \odot H$ a graph F by reducing each H-layer composed of non-simplicial vertices to a single vertex. By Lemma 82, $\text{hn}(G \odot H) = \text{hn}(F)$. Then, we apply Lemma 84 to reduce each H-layer of simplicial vertices to a single vertex obtaining a graph F'. Observe that we have $|V(H)||S(G)|$ simplicial vertices in $G \odot H$ and, thus,

$(|V(H)| - 1)|S(G)|$ identifications are done in this procedure. Finally, since all the H-layers were reduced to a single vertex, observe that F' is isomorphic to G and we have that $\mathrm{hn}(G \odot H) = \mathrm{hn}(F) = \mathrm{hn}(F') + (|V(H)| - 1)|S(G)| = \mathrm{hn}(G) + (|V(H)| - 1)|S(G)|$. □

9.7 Hull Number via Two Connected Components

In this section, we introduce the generalized hull number of a graph. Let $G = (V, E)$ be a graph and $S \subseteq V$. The *generalized hull number*, denoted by $\mathrm{hn}(G, S)$, is the minimum size of a set $U \subseteq V \setminus S$ such that $U \cup S$ is a hull set for G. We prove that to compute the hull number of a graph, it is sufficient to compute the generalized hull number of its 2-connected components (or *blocks*). This extends a result in [ES85].

Theorem 87. *Let G be a graph and $G_1, \ldots, G_n$ be its 2-connected components. For any $i \leq n$, let $S_i \subseteq V(G_i)$ be the set of cut-vertices of G in G_i. Then,*

$$\mathrm{hn}(G) = \sum_{i \leq n} \mathrm{hn}(G_i, S_i).$$

Proof. Clearly, the result holds if $n = 1$, so we assume $n > 1$.

A block G_i is called a *leaf-block* if $|S_i| = 1$. Note that, for any leaf-block G_i, $G[V \setminus (V(G_i) \setminus S_i)]$ is convex, so by Lemma 66, any hull set of G contains at least one vertex in $V(G_i) \setminus S_i$. Moreover, for any minimum hull set S of G, $S \cap (\bigcup_{i \leq n} S_i) = \emptyset$. To prove this fact, it is sufficient to observe that, for any cut-vertex v, there exist two vertices u and v in disjoint leaf-blocks such that v in a shortest (u, v)-path.

Claim 24. *Let S be a hull set of G. Then $S' = (S \cap V(G_i)) \cup S_i$ is a hull set of G_i.*

Claim 25. *For purpose of contradiction, assume that $I_h[S'] = V(G_i) \setminus X$ for some $X \neq \emptyset$. Then, there is $v \in X \cap I[a, b]$ for some $a \in V(G) \setminus V(G_i)$ and $b \in V(G) \setminus X$. Then, there is a shortest (a, b)-path P containing v. Hence, there is $u \in S_i$ such that u is on the subpath of P between a and v. Moreover, let $w = b$ if $b \in G_i$, and else let w be a vertex of S_i on the subpath of P between v and b. Hence, $v \in I[u, w] \subseteq I_h[S']$, a contradiction.*

Let X be any minimum hull set of G. Since, $X \cap (\bigcup_{i \leq n} S_i) = \emptyset$, hence we can partition $X = \bigcup_{i \leq n} X_i$ such that $X_i \subseteq V(G_i) \setminus S_i$ and $X_i \cap X_j = \emptyset$ for any $i \neq j$. Moreover, by Claim 24, $X_i \cup S_i$ is a hull set of G_i, i.e., $|X_i| \geq \mathrm{hn}(G_i, S_i)$. Hence, $\mathrm{hn}(G) = |X| = \sum_{i \leq n} |X_i| \geq \sum_{i \leq n} \mathrm{hn}(G_i, S_i)$.

It remains to prove the reverse inequality. For any $i \leq n$, let $X_i \subseteq V(G_i) \setminus S_i$ such that $X_i \cup S_i$ is a hull set of G_i and $|X_i| = \mathrm{hn}(G_i, S_i)$. We prove that $S = \cup_{i \leq n} X_i$ is a hull set for G. Indeed, for any $v \in S_i$, there are two leaf-blocks G_1, G_2 such that v is on a shortest path between G_1 and G_2 or $\{v\} = V(G_1) \cap V(G_2)$. So, there exist $x \in X_1$ and $y \in X_2$ such that v is on a shortest (x, y)-path, i.e., $v \in I[x, y] \subseteq I_h[S]$. Hence, $\bigcup_{i \leq n} S_i \subseteq I_h[S]$ and therefore, $V = \bigcup_{i \leq n} I_h[X_i \cup S_i] \subseteq I_h[\bigcup_{i \leq n}(X_i \cup S_i)] \subseteq I_h[\bigcup_{i \leq n}(X_i)] = I_h[S]$. □

A *cactus* G is a graph in which every pair of cycles have at most one common vertex. This definition implies that each block of G is either a cycle or an edge. By using the previous result, one may easily prove that:

Corollary 8. *In the class of cactus graphs, the hull number can be computed in linear time.*

9.8 Bounds For the Hull Number of Graphs

In this section, we use the same techniques as presented in [ES85] and in [DPRS10a] to prove new bounds on the hull number of several graph classes. These techniques mainly rely on a greedy algorithm for computing a hull set of a graph and that consists of the following: given a connected graph $G = (V, E)$ and its set S of simplicial vertices, we start with $H = S$ or $H = \{v\}$ (v is any vertex of V) if $S = \emptyset$, and $C_0 = I_h[H]$. Then, at each step $i \geq 1$, if $C_{i-1} \subset V$, the algorithm greedily chooses a subset $X_i \subseteq V \setminus C_{i-1}$, add X_i to H and set $C_i = I_h[H]$. Finally, if $C_i = V$, the algorithm returns H which is a hull set of G.

Claim 26. *If for every $i \geq 1$, $|C_i \setminus (C_{i-1} \cup X_i)| \geq c|X_i|$, for some constant $c > 0$, then*

$$|H| \leq \max\{1, |S|\} + \left\lceil \frac{|V| - \max\{1, |S|\}}{1 + c} \right\rceil.$$

For the rest of this section, we follow the notation used to describe the algorithm in the beginning of the section.

Claim 27. *Let G be a connected graph. Then, before each step $i \geq 1$ of the algorithm, for any $v \in V \setminus C_{i-1}$, $N(v) \cap C_{i-1}$ induces a clique. Moreover, any connected components induced by $V \setminus C_{i-1}$ has at least 2 vertices.*

Proof. Let $v \in V \setminus C_{i-1}$ and assume v has two neighbors u and w in C_{i-1} that are not adjacent. Then, $v \in I[u, w] \subseteq C_{i-1}$ because C_{i-1} is convex, a contradiction. Note that, at any step $i \geq 1$ of the algorithm, $V \setminus C_{i-1}$ contains no simplicial vertex. By previous remark, if v has only neighbors in C_{i-1}, then v is simplicial, a contradiction. □

Claim 28. *If G is a connected C_3-free graph, then, at every step $i \geq 1$ of the algorithm, a vertex in $V \setminus C_{i-1}$ has at most one neighbor in C_{i-1}.*

Proof. Assume that $v \in V \setminus C_{i-1}$ has two neighbors $u, w \in C_{i-1}$. $\{u, w\} \notin E$ because G is triangle-free. This contradicts Claim 27. □

Lemma 88. *For any C_3-free connected graph G and at step $i \geq 1$ of the algorithm, either $C_{i-1} = V$ or there exists $X_i \subset V \setminus C_{i-1}$ such that $|C_i \setminus (C_{i-1} \cup X_i)| \geq |X_i|$.*

Proof. If there is $v \in V \setminus C_{i-1}$ at distance at least 2 from C_{i-1}, let $X_i = \{v\}$ and the result clearly holds. Otherwise, let v be any vertex in $V \setminus C_{i-1}$. By Claim 27, v has a neighbor u in $V \setminus C_{i-1}$. Moreover, because no vertices of $V \setminus C_{i-1}$ are at distance at least 2 from C_{i-1}, v and u have some neighbors in C_{i-1}. Finally, u and v have no common neighbors because G is triangle-free. Hence, by taking $X_i = \{v\}$, we have $u \in C_i$ and the result holds. □

Recall that the *girth* of a graph is the length of its smallest cycle.

Lemma 89. *Let G connected with girth at least 6. Before any step $i \geq 1$ of the algorithm when $C_{i-1} \neq V$, there exists $X_i \subset V \setminus C_{i-1}$ such that $|C_i \setminus (C_{i-1} \cup X_i)| \geq 2|X_i|$.*

Proof. If there is $v \in V \setminus C_{i-1}$ at distance at least 3 from C_{i-1}, let $X_i = \{v\}$ and the result clearly holds. Otherwise, let v be a vertex in $V \setminus C_{i-1}$ at distance two from any vertex of C_{i-1}. Let $w \in V \setminus C_{i-1}$ be a neighbor of v that has a neighbor $z \in C_{i-1}$. Since v is not simplicial, v has another neighbor $u \neq w$ in $V \setminus C_{i-1}$. If u is at distance two

from C_{i-1}, let $y \in V \setminus C_{i-1}$ be a neighbor of u that has a neighbor $x \in C_{i-1}$. In this case, since the girth of G is at least six, $z \neq x$ and, there is a shortest (v, z)-path containing w and a shortest (v, x)-path containing u and y. Consequently, by setting $X_i = \{v\}$ we obtain the desired result. The same happens in case u has a neighbor $x \in C_{i-1}$. One may use again the hypothesis that the girth of G is at least six to conclude that, by setting $X_i = \{v\}$ we obtain that $w, u \in C_i$.

Finally, we claim that no vertex remains in $V \setminus C_{i-1}$. By contradiction, suppose that it is the case and that there are in $V \setminus C_{i-1}$ and all these vertices have a neighbor in C_{i-1}. Let v be a vertex in $V \setminus C_{i-1}$ that has a neighbor z in C_{i-1}. Again, v has a neighbor $u \in V \setminus C_{i-1}$, since it is not simplicial. The vertex u must have a neighbor x in C_{i-1}. Observe that x and z are at distance 3, since the girth of G is at least six. Consequently, v and u are in a shortest (x, z)-path should not be in $V \setminus C_{i-1}$, that is a contradiction. □

Lemma 90. *Let G be a connected graph. Before any step $i \geq 1$ of the algorithm when $C_{i-1} \neq V$, there exist $X_i \subset V \setminus C_{i-1}$ such that $|C_i \setminus (C_{i-1} \cup X_i)| \geq 2|X_i|/3$.*

Moreover, if G is k-regular ($k \geq 1$), there exist $X_i \subset V \setminus C_{i-1}$ such that $|C_i \setminus (C_{i-1} \cup X_i)| \geq |X_i|$.

Proof. By Claim 27, all connected component of $V \setminus C_{i-1}$ contains at least one edge.

- If there is $v \in V \setminus C_{i-1}$ at distance at least 2 from C_{i-1}, let $X_i = \{v\}$ and $|C_i \setminus (C_{i-1} \cup X_i)| \geq |X_i|$.

- Now, assume all vertices in $V \setminus C_{i-1}$ are adjacent to some vertex in C_{i-1}. If there are two adjacent vertices u and v in $V \setminus C_{i-1}$ such that there is $z \in C_{i-1} \cap N(u) \setminus N(v)$, then let $X_i = \{v\}$. Therefore, $u \in C_i$ and $|C_i \setminus (C_{i-1} \cup X_i)| \geq |X_i|$. So, the result holds.

- Finally, assume that for any two adjacent vertices u and v in $V \setminus C_{i-1}$, $N(u) \cap C_{i-1} = N(v) \cap C_{i-1} \neq \emptyset$.

  We first prove that this case actually cannot occur if G is k-regular. Let $v \in V \setminus C_{i-1}$. By Claim 27, $K = N(v) \cap C_{i-1}$ induces a clique. Moreover, for any $u \in N(v) \setminus C_{i-1}$, $N(u) \cap C_{i-1} = K$. Note that $k = |K| + |N(v) \setminus C_{i-1}|$. Let $w \in K$. Then, $A = (K \cup N(v) \cup \{v\}) \setminus \{w\} \subseteq N(w)$ and since $|A| = k$, we get that $A = N(w)$. Moreover, $N[u]$ cannot induce a clique since $V \setminus C_{i-1}$ contains no simplicial vertices, $i \geq 1$. Hence, there are $x, y \in N(v) \setminus C_{i-1}$ such that $\{x, y\} \notin E$. Because G is k-regular, there is $z \in N(x) \setminus (N(v) \cup C_{i-1})$. However, $N(z) \cap C_{i-1} = N(x) \cap C_{i-1} = K$. Hence, $z \in N(w) \setminus A$, a contradiction.

  Now, assume that G is a general graph. Let v be a vertex of minimum degree in $V \setminus C_{i-1}$. Recall that, by Claim 27, $N(v) \cap C_{i-1}$ induces a clique. Because any neighbor $u \in V \setminus C_{i-1}$ of v has the same neighborhood as v in C_{i-1} and because v is not simplicial, then there must be $u, w \in N(v) \setminus C_{i-1}$ such that $\{u, w\} \notin E$. Now, by minimality of the degree of v, there exists $y \in N(u) \setminus (N(v) \cup C_{i-1}) \neq \emptyset$. Similarly, there exists $z \in N(w) \setminus (N(v) \cup C_{i-1}) \neq \emptyset$. Let us set $X_i = \{v, z, y\}$. Hence, $u, w \in C_i \setminus (C_{i-1} \cup X_i)$ and the result holds.

□

Theorem 91. *Let G be a connected n-node graph with s simplicial vertices. All bounds below are tight:*

$$\mathrm{hn}(G) \leq \max\{1, s\} + \left\lceil \frac{3(n - \max\{1, s\})}{5} \right\rceil;$$

If G is C_3-free or k-regular ($k \geq 1$), then

$$\mathrm{hn}(G) \leq \max\{1, s\} + \left\lceil \frac{n - \max\{1, s\}}{2} \right\rceil;$$

If G has girth at least 6, then

$$\mathrm{hn}(G) \leq \max\{1, s\} + \left\lceil \frac{1(n - \max\{1, s\})}{3} \right\rceil.$$

Proof. The first statement follows from Claim 26 and first statement in Lemma 90. The second statement follows from Claim 26 and Lemma 88 (the case where G is C_3-free) and the second part of Lemma 90 (the case of regular graphs). The last statement follows from Claim 26 and Lemma 89.

All bounds are reached in the case of complete graphs. In case with no simplicial vertices: the first bound is reached by the graph obtained by taking several disjoint C_5 and adding a universal vertex, the second bound is obtained for a C_5, and the third one is reached by a C_7. □

The first statement of the previous theorem improves another result in [ES85]:

Corollary 9. *If G is a graph with no simplicial vertex, then*

$$\limsup_{|V(G)| \to \infty} \frac{\mathrm{hn}(G)}{|V(G)|} = \frac{3}{5}.$$

It it important to remark that the second statement of Theorem 91 is closely related to a bound of Everett and Seidman proved in Theorem 9 of [ES85]. However, the graphs they consider do not have simplicial vertices and, consequently, they do not have vertices of degree one, which is not a constraint for our result.

9.9 Conclusion

In this chapter, we simplified the reduction of Dourado *et al.* [DGK+09] to answer a question they asked about the complexity of computing the hull number of bipartite graphs. Then, we presented polynomial-time algorithms for computing the hull number of co-bipartite graphs, $(q, q-4)$-graphs, cactus graphs and $\{P_5, K_3\}$-free graphs.

In particular, the complexity of the algorithm proposed to compute the hull number of any P_5-free triangle-free graph is linear on the size of the input graph. However, the computational complexity of determining the hull number of a P_5-free graph is still unknown. More generally, we propose the following open question: for a fixed k, what is the computational complexity of determining $\mathrm{hn}(G)$, for a P_k-free graph G?

The algorithm presented in Section 9.4 is an FPT algorithm to compute the hull number of any graph where the parameter is the number of its induced P_4's. Then, in Section 9.6, we introduced three reduction rules that we use to construct an FPT algorithm to compute the hull number of any graph, where the parameter is its neighborhood

diversity, and a characterization of the lexicographic product of any two graphs. However, the parameters of both of these algorithms do not seem to have a direct relationship with the hull number. Hence, we also propose the following question: given a graph G, is there an FPT algorithm to determine whether $\mathrm{hn}(G) \leq k$, for a fixed k?

Finally, in Section 9.8, we presented some upper bounds for the hull number of general graphs and for some graph classes. Albeit the proofs for the bounds proposed in this chapter use the same techniques as the ones in [DPRS10a], they are not comparable between themselves.

Chapter 10

Conclusion

In this thesis, we studied three fundamental subjects in Graph Theory, namely, graph decompositions, pursuit-evasion games and convexity. In this chapter, we reiterate some important results of this thesis with a discussion about their shortcomings and perspectives of future work.

In Chapter 3, the first result presented was the monotonicity of the *Process game*. This property allowed us to design a directed graph decomposition, the *Process decomposition*, that is "equivalent" to the *Process game*. We proved that, for every directed graph G, if the directed path width of G is at most k, then there is a process decomposition $((W_1, X_1), \ldots, (W_n, X_n))$ of G with width k such that every bag X_i is a singleton. One consequence of this result is that, $(W_1 \cup X_1, \ldots, W_n \cup X_n)$ is a directed path decomposition of G with width at most $k+1$. By consequence, the *Process decomposition* has a close relationship with the *Directed Path Decomposition*. This also means that, for every directed graph G, we have that $\mathrm{dpw}(G) - 1 \leq \mathrm{pw}(G) \leq \mathrm{dpw}(G)$, once every path decomposition is a *Process decomposition* where bags inducing DAGs are empty. The *DAG decompositions* and the *Directed path decomposition* are related with the *visible Directed Node Search* and the *invisible Directed Node Search* respectively. Since *visible Directed Node Search* is not monotone, while *invisible Directed Node Search* is monotone, we wonder if the *Process game* is still monotone when the robber is visible. If it is, it could help us better understand what makes a directed graph searching game monotone.

In Chapter 4, we investigated the problem of computing several width measures of graphs. We proposed an FPT-algorithm with parameter k to decide if a graph has width at most k. This algorithm can be applied to several widths including the *special tree width* and the *q-branched tree width* for which no previous explicit algorithm was known. Since it can be used with any width, that can be represented in terms of partition functions and partitioning trees satisfying some restrictions explained in this chapter, we wonder if there are other width measures that can be computed by this algorithm. Furthermore, we also wonder if we could extend these results for directed graph decomposition, by having a common representation of different widths by some kind of directed partitioning tree.

The second part of this thesis focused on turn-by-turn pursuit-evasion games. In Chapter 6, we studied the cost of requiring connectivity on the *Surveillance game*, that is, we studied how big can be the difference between the *connected surveillance number* and the *surveillance number*. In despite of the fact that we have improved a previous result of Fomin *et al.* [FGJM+12], by showing that there are graphs where this difference

is of two, we were unable to construct any graph where this difference is at at least three. This leads us to believe that their conjecture of this difference being at most a constant is true. Moreover, we also showed the first upper bound, that is not trivial, for the cost of connectivity. That is, we showed that, for any graph G, $\operatorname{csn}(G) \leq \sqrt{|V(G)| \operatorname{sn}(G)}$, which is still far from having a constant addictive factor.

In Chapter 6, we also defined the *Online Surveillance game*. In this game, the observer discovers the graph, on which the game is played, little-by-little, in an attempt to better model the web-page prefetching problem which originally motivated the *Surveillance game*. We show that there, for any $k \in \mathbb{N}$, there exists a tree T_k such that $\operatorname{on}(T_k) = O(\Delta)$ while $\operatorname{sn}(T_k) = 2$. This is unfortunate for two reasons. The first one being that the best strategy for the observer, up to constant factors, is to simply mark the neighborhood of the current vertex occupied by the surfer. The second one is that the difference between the *Online surveillance number* and the *connected surveillance number* might be arbitrarily large, since, in trees, the *connected surveillance number* is equal to the *surveillance number*. This implies that the *Online Surveillance game* is not a good candidate in order to study the cost of enforcing connectivity.

In Chapter 7, we studied fractional turn-by-turn pursuit-evasion games. We defined a framework for turn-by-turn pursuit-evasion games, in which tokens controlled by players can be split into fractions, that can model several turn-by-turn pursuit-evasion games. Based on this framework, we proposed an algorithm, that uses linear programming techniques, which decides if there is a winning strategy for the pursuer for any game modeled by this framework. On the bright side, several games such as the *Cops and Robbers*, *Surveillance game*, *Angels problem*, *Eternal Dominating Set* and *Eternal Vertex Cover* fit this framework and, thus, can be solved by our algorithm. However, the complexity of the algorithm proposed is exponential on the size of the input graph due to a step where we might create an exponential number of inequalities. Despite this, it seems that the number of redundant inequalities created in this step is rather large, implying that we might be able to construct a polynomial time algorithm with a closer analysis of how these inequalities are created.

In Chapter 7, we also showed that a fractional game's parameter is a lower bound for the correspondent integral game's parameter. Then, to answer how big can the gap be between these two parameters, we focused on particular turn-by-turn pursuit-evasion games. We showed that the number of fractional cops necessary to capture a fractional robber, in a number of turns linear on the number of vertices, is $1+\epsilon$, $\epsilon > 0$. However, such strategy can only be applied if the robber has speed one, hence, it might be interesting to investigate the *fractional Cops and Robbers* when the robber has speed bigger than 1 in hopes that this might leads us to new insights into solving the Meyniel's conjecture for a faster robber. We also proved that winning strategies for fractional *Surveillance game* and for the fractional *Angels problem* game can be used to help the pursuer to win in the integral versions of these games.

Finally, the third part of this thesis was dedicated to the study of the hardness of computing the hull number in several graph classes. We first showed that the decision problem associated with the hull number of a bipartite graph is NP-complete, successfully answering an open question in [DGK$^+$09]. Additionally, we showed that there are polynomial time algorithms for computing the hull number of a co-bipartite graph, $(q, q-4)$ graphs and $\{P_5, K_3\}$-free graph. The main technique used in the algorithms for co-bipartite graphs and for $\{P_5, K_3\}$-free graphs was to exclude some structure, either an edge or some vertices, and then decompose the remaining graph into connected components. We

wonder if such approach might be useful when applied to other classes of graphs. On the other hand, the algorithm for $(q, q-4)$-graphs, which is a superclass of P_4-sparse graphs, was mainly based on a dynamic programming approach guided by a modular decomposition of such graphs. Since all these algorithms use some kind of decomposition of the graph, we wonder if there is a more suitable decomposition for computing the hull number of graphs.

Another important result of Chapter 9 is an FPT-algorithm for computing the hull number of general graphs, where the parameter can be either the neighborhood diversity or the minimum vertex cover number of the input graph. However, since these parameters do not seem to have a direct relation with the hull number, it would be interesting to have an FPT-algorithm for computing the hull number where the parameter is the hull number itself.

Future Work. In this thesis, we proposed several algorithms for solving different problems in the subjects of pursuit-evasion games, graph decompositions and convexity. Although the FPT-algorithm proposed for computing width measures of graphs might be impractical even for small parameters, we are interested in measuring the performance of the other algorithms proposed in this thesis in practice. We are also interested in investigating the monotonicity of a visible fugitive in the *Process game*. However, it is unlikely that this property holds, since the visible directed graph searching games studied in the literature lack this property. Finally, we aim at investigating the complexity of *fractional turn-by-turn pursuit-evasion games*.

Index

Bibliography

[ACG+11] J. Araujo, V. Campos, F. Giroire, L. Sampaio, and R. Soares. On the hull number of some graph classes. *Electronic Notes in Discrete Mathematics*, 38(0):49–55, 2011. 7

[ACG+13] J. Araujo, V. Campos, F. Giroire, L. Sampaio, and R. Soares. On the hull number of some graph classes. *Theoretical Computer Science*, 475(0):1–12, 2013. 7

[ACKP12] B.S. Anand, M. Changat, S. Klavžar, and I. Peterin. Convex sets in lexicographic products of graphs. *Graphs and Combinatorics*, 28(1):77–84, 2012. 151

[ACP87] S. Arnborg, D. Corneil, and A. Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM Journal on Algebraic Discrete Methods*, 8(2):277–284, 1987. 3, 14, 15

[Adl07] I. Adler. Directed tree-width examples. *Journal of Combinatorial Theory, Series B*, 97(5):718–725, 2007. 4, 24

[AEFP98] Y. Aumann, O. Etzioni, R. Feldman, and M. Perkowitz. Predicting event sequences: Data mining for prefetching web-pages, 1998. 111

[AF84] M. Aigner and M. Fromme. A game of cops and robbers. *Discrete Applied Mathematics*, 8:1–12, 1984. 102, 103, 104

[Als04] B. Alspach. Searching and sweeping graphs: a brief survey. *Le Matematiche*, 59(1-2), 2004. 4

[AMNT09] O. Amini, F. Mazoit, N. Nisse, and S. Thomassé. Submodular partition functions. *Discrete Mathematics*, 309(20):6000–6008, 2009. 43, 44, 46, 47, 98

[AMS+13] J. Araujo, G. Morel, L. Sampaio, R. Soares, and V. Weber. Hull number: p_5-free graphs and reduction rules. In *LAGOS*, 2013. 7

[AP89] S. Arnborg and A. Proskurowski. Linear time algorithms for np-hard problems restricted to partial k-trees. *Discrete Applied Mathematics*, 23(1):11–24, 1989. 2, 3, 14

[AZN99] D. Albrecht, I. Zukerman, and A. Nicholson. Pre-sending documents on the www: a comparative study. In *Proceedings of the 16th international joint conference on Artificial intelligence - Volume 2*, IJCAI'99, pages 1274–1279, 1999. 111

[Bar06] J. Barát. Directed path-width and monotonicity in digraph searching. *Graphs and Combinatorics*, 22(2):161–172, 2006. 4, 23, 26, 27, 32, 33, 34

[BB12] W. Baird and A. Bonato. Meyniel's conjecture on the cop number: A survey. *Journal of Combinatorics*, 3(2):225–238, 2012. 103

[BCG+04] A. P. Burger, E. J. Cockayne, W. R. Ggündlingh, C. M. Mynhardt, J. H. Van Vuuren, and W. Winterbach. Infinite order domination in graphs. *Journal of Combinatorial Mathematics and Combinatorial Computing*, 50:179–194, 2004. 106

[BDFM12] A. Beveridge, A. Dudek, A. Frieze, and T. Müller. Cops and robbers on geometric graphs. *Combinatorics, Probability and Computing*, 21:816–834, 10 2012. 104

[BDH+12] D. Berwanger, A. Dawar, P. Hunter, S. Kreutzer, and J. Obdržálek. The dag-width of directed graphs. *Journal of Combinatorial Theory, Series B*, 102(4):900–923, 2012. 23, 25, 27

[BFST03] L. Barrière, P. Fraigniaud, N. Santoro, and D.M. Thilikos. Connected and internal graph searching. In *In 29th Workshop on Graph Theoretic Concepts in Computer Science (WG), LNCS 2880*, pages 34–45, 2003. 23

[BGHK95] H.L. Bodlaender, J.R. Gilbert, H. Hafsteinsson, and T. Kloks. Approximating treewidth, pathwidth, frontsize, and shortest elimination tree. *Journal of Algorithms*, 18(2):238–255, 1995. 15

[BGHK09] A. Bonato, P. Golovach, G. Hahn, and J. Kratochvíl. The capture time of a graph. *Discrete Mathematics*, 309(18):5588–5595, 2009. 105

[BK96] H.L. Bodlaender and T. Kloks. Efficient and constructive algorithms for the pathwidth and treewidth of graphs. *Journal of Algorithms*, 21(2):358–402, 1996. 18, 48, 50, 51, 59, 60, 98

[BKL13] B. Bollobás, G. Kun, and I. Leader. Cops and robbers in a random graph. *Journal of Combinatorial Theory, Series B*, 103(2):226–236, 2013. 103, 104

[BKT08] B. Brešar, S. Klavžar, and A.H. Tepeh. On the geodetic number and related metric sets in cartesian product graphs. *Discrete Mathematics*, 308(23):5555–5561, 2008. 151

[BKT11] B. Brešar, M. Kovše, and A. Tepeh. Geodetic sets in graphs. In *Structural Analysis of Complex Networks*, pages 197–218. Birkhäuser Boston, 2011. 151

[BL83] L. Babai and E.M. Luks. Canonical labeling of graphs. In *Proceedings of the fifteenth annual ACM symposium on Theory of computing*, pages 171–183. ACM, 1983. 95

[BL06] B. Bollobás and I. Leader. The angel and the devil in three dimensions. *Journal of Combinatorial Theory, Series A*, 113(1):176–184, 2006. 107

[BM08] J.A. Bondy and U.S.R. Murty. *Graph Theory*. Graduate texts in mathematics. Springer, 2008. 8

[BN11] A. Bonato and R.J. Nowakowski. *The game of cops and robbers on graphs*. Student mathematical library. American Mathematical Society, 2011. 105

[BO99] L. Babel and S. Olariu. On the p-connectedness of graphs - a survey. *Discrete Applied Mathematics*, 95(1-3):11–33, 1999. 162

[Bod96] H.L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing*, 25(6):1305–1317, 1996. 15, 43, 44, 48, 50, 62

[Bod98] H.L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theoretical Computer Science*, 209(1–2):1–45, 1998. 14, 19

[Bow07] B.H. Bowditch. The angel game in the plane. *Combinatorics, Probability and Computing*, 16(3):345–362, 2007. 107

[BPS10] S. Bhattacharya, G. Paul, and S. Sanyal. A cops and robber game in multidimensional grids. *Discrete Applied Mathematics*, 158(16):1745–1751, 2010. 104

[BST00] H.L. Bodlaender, M.J. Serna, and D.M. Thilikos. Constructive linear time algorithms for small cutwidth and carving-width. In *Algorithms and Computation*, volume 1969 of *Lecture Notes in Computer Science*, pages 192–203. Springer Berlin Heidelberg, 2000. 19

[BT90] G. Bacsó and Z. Tuza. Dominating cliques in p_5-free graphs. *Periodica Mathematica Hungarica*, 21(4):303–308, 1990. 167

[BT97] H.L. Bodlaender and D.M. Thilikos. Constructive linear time algorithms for branchwidth. In *Automata, Languages and Programming*, volume 1256 of *Lecture Notes in Computer Science*, pages 627–637. Springer Berlin Heidelberg, 1997. 16, 18, 43, 47, 48

[BT04] H.L. Bodlaender and D.M. Thilikos. Computing small search numbers in linear time. In *Parameterized and Exact Computation*, volume 3162 of *Lecture Notes in Computer Science*, pages 37–48. Springer Berlin Heidelberg, 2004. 18, 19, 43, 47, 48

[Cay89] A. Cayley. A theorem on trees. *Quarterly Journal of Mathematics*, 23:376–378, 1889. 71

[CCJ04] G.B. Cagaanan and S.R. Canoy Jr. On the hull sets and hull number of the cartesian product of graphs. *Discrete Mathematics*, 287(1–3):141–144, 2004. 151

[CCM+11] N. Cohen, D. Coudert, D. Mazauric, N. Nepomuceno, and N. Nisse. Tradeoffs in process strategy games with application in the wdm reconfiguration problem. *Theoretical Computer Science*, 412(35):4675–4687, 2011. 30, 31

[CCNV11] J. Chalopin, V. Chepoi, N. Nisse, and Y. Vaxès. Cop and robber games when the robber can hide and ride. *SIAM Journal on Discrete Mathematics*, 25(1):333–359, 2011. 104

[CDD+10] C.C. Centeno, S. Dantas, M.C. Dourado, D. Rautenbach, and J.L. Szwarcfiter. Convex partitions of graphs induced by paths of order three. *Discrete Mathematics & Theoretical Computer Science*, 12(5):175–184, 2010. 5, 149

[CDP+11] C.C. Centeno, M.C. Dourado, L.D. Penso, D. Rautenbach, and J.L. Szwarcfiter. Irreversible conversion of graphs. *Theoretical Computer Science*, 412(29):3693–3700, 2011. 148, 149

[CDS09] C.C. Centeno, M.C. Dourado, and J.L. Szwarcfiter. On the convexity of paths of length two in undirected graphs. *Electronic Notes in Discrete Mathematics*, 32(0):11–18, 2009. 147, 149

[Chi08] E. Chiniforooshan. A better bound for the cop number of general graphs. *Journal of Graph Theory*, 58(1):45–48, 2008. 103

[CHM+09] D. Coudert, F. Huc, D. Mazauric, N. Nisse, and J.-S. Sereni. Reconfiguration of the routing in wdm networks with two classes of services. In *Proceedings of the 13th international conference on Optical Network Design and Modeling*, ONDM'09, pages 146–151, 2009. 31

[CHM+10] J. Cáceres, C. Hernando, M. Mora, I.M. Pelayo, and M.L. Puertas. On the geodetic and the hull numbers in strong product graphs. *Computers & Mathematics with Applications*, 60(11):3020–3031, 2010. 151

[CHM12] D. Coudert, F. Huc, and D. Mazauric. A distributed algorithm for computing the node search number in trees. *Algorithmica*, 63(1-2):158–190, 2012. 31, 32

[CJG02] S.R. Canoy Jr. and I.J.L. Garces. Convex sets under some graph operations. *Graphs and Combinatorics*, 18(4):787–793, 2002. 151

[Cla02] N.E. Clarke. *Constrained Cops and Robber*. PhD thesis, Dalhousie University, 2002. 104

[CMS05] M. Changat, H.M. Mulder, and G. Sierksma. Convexities related to path properties on graphs. *Discrete Mathematics*, 290(2–3):117–131, 2005. 151

[Con98] J.H. Conway. The angel problem. In R.J. Nowakowski, editor, *Games of No Chance*, volume 29 of *Mathematical Sciences Research Institute Publications*, pages 3–12. Cambridge University Press, 1998. 107

[Cou89] B. Courcelle. The monadic second-order logic of graphs : Definable sets of finite graphs. In *Graph-Theoretic Concepts in Computer Science*, volume 344 of *Lecture Notes in Computer Science*, pages 30–53. Springer Berlin Heidelberg, 1989. 2, 14, 15

[Cou10] B. Courcelle. Special tree-width and the verification of monadic second-order graph properties. In *FSTTCS*, pages 13–29, 2010. 15, 19, 43

[CPPS05] D. Coudert, S. Perennes, Q-C. Pham, and J-S. Sereni. Rerouting requests in wdm networks. In *7ème Rencontres Francophones sur les Aspects Algorithmiques des Télécommunications (AlgoTel'05)*, pages 17–20, Presqu'île de Giens, France, 2005. 4, 29, 30, 31

[CRST06] M. Chudnovsky, N. Robertson, P. Seymour, and R. Thomas. The strong perfect graph theorem. *Annals of Mathematics*, 164:51–229, 2006. 3

[CS03] W. Cook and P. Seymour. Tour merging via branch-decomposition. *INFORMS Journal on Computing*, 15(3):233–248, 2003. 2, 17

[CS11] D. Coudert and J-S. Sereni. Characterization of graphs and digraphs with small process number. *Discrete Applied Mathematics*, 159(11):1094–1109, 2011. 31

[CWZ02] G. Chartrand, C.E. Wall, and P. Zhang. The convexity number of a graph. *Graphs and Combinatorics*, 18(2):209–217, 2002. 146, 150

[DGK+09] M.C. Dourado, J.G. Gimbel, J. Kratochvíl, F. Protti, and J.L. Szwarcfiter. On the computation of the hull number of a graph. *Discrete Mathematics*, 309(18):5668–5674, 2009. 6, 7, 148, 149, 153, 154, 155, 161, 164, 170, 177, 180

[DJR09] P.A. Dreyer Jr. and F.S. Roberts. Irreversible -threshold processes: Graph-theoretical threshold models of the spread of disease and of opinion. *Discrete Applied Mathematics*, 157(7):1615–1627, 2009. 149

[DKT97] N.D. Dendris, L.M. Kirousis, and D.M. Thilikos. Fugitive-search games on graphs and related parameters. *Theoretical Computer Science*, 172(1–2):233–254, 1997. 26

[DPRS10a] M. Dourado, F. Protti, D. Rautenbach, and J. Szwarcfiter. On the hull number of triangle-free graphs. *SIAM Journal on Discrete Mathematics*, 23(4):2163–2172, 2010. 150, 175, 178

[DPRS10b] M.C. Dourado, F. Protti, D. Rautenbach, and J.L. Szwarcfiter. Some remarks on the geodetic number of a graph. *Discrete Mathematics*, 310(4):832–837, 2010. 148, 149, 150

[DPRS12] M.C. Dourado, F. Protti, D. Rautenbach, and J.L. Szwarcfiter. On the convexity number of graphs. *Graphs and Combinatorics*, 28(3):333–345, 2012. 148, 149

[DPS10] M.C Dourado, F. Protti, and J.L. Szwarcfiter. Complexity results related to monophonic convexity. *Discrete Applied Mathematics*, 158(12):1268–1274, 2010. 148, 149

[ES85] M.G. Everett and S.B. Seidman. The hull number of a graph. *Discrete Mathematics*, 57(3):217–223, 1985. 154, 174, 175, 177

[FCLJ99] L. Fan, P. Cao, W. Lin, and Q. Jacobson. Web prefetching between low-bandwidth clients and proxies: potential and performance. In *Proceedings of the 1999 ACM SIGMETRICS international conference on Measurement*

and modeling of computer systems, SIGMETRICS '99, pages 178–187, 1999. 111

[FFN05] F.V. Fomin, P. Fraigniaud, and N. Nisse. Nondeterministic graph searching: From pathwidth to treewidth. In *Mathematical Foundations of Computer Science 2005*, volume 3618 of *Lecture Notes in Computer Science*, pages 364–375. Springer Berlin Heidelberg, 2005. 16

[FGG+10] F.V. Fomin, S. Gaspers, P.A. Golovach, D. Kratsch, and S. Saurabh. Parameterized algorithm for eternal vertex cover. *Information Processing Letters*, 110(16):702–706, 2010. 106

[FGJM+12] F.V. Fomin, F. Giroire, A. Jean-Marie, D. Mazauric, and N. Nisse. To satisfy impatient web surfers is hard. In *Fun with Algorithms*, volume 7288 of *Lecture Notes in Computer Science*, pages 166–176. Springer Berlin Heidelberg, 2012. 4, 109, 110, 111, 112, 113, 119, 125, 179

[FGK+10] F.V. Fomin, P.A. Golovach, J. Kratochvíl, N. Nisse, and K. Suchan. Pursuing a fast robber on a graph. *Theoretical Computer Science*, 411(7–9):1167–1181, 2010. 103, 104

[FGL12] F.V. Fomin, P.A. Golovach, and D. Lokshtanov. Cops and robber game without recharging. *Theory of Computing Systems*, 50(4):611–620, 2012. 105

[FGP12] F.V. Fomin, P.A. Golovach, and P. Pralat. Cops and robber with constraints. *SIAM Journal on Discrete Mathematics*, 26(2):571–590, 2012. 4, 105

[FHL05] U. Feige, M.T. Hajiaghayi, and J.R. Lee. Improved approximation algorithms for minimum-weight vertex separators. In *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pages 563–572, 2005. 15

[FJ86] M. Farber and R.E. Jamison. Convexity in graphs and hypergraphs. *SIAM Journal on Algebraic and Discrete Methods*, 7(3):433–444, 1986. 145, 146

[FKL12] A. Frieze, M. Krivelevich, and P. Loh. Variations on cops and robbers. *Journal of Graph Theory*, 69(4):383–402, 2012. 103

[FN08] P. Fraigniaud and N. Nisse. Monotony properties of connected visible graph searching. *Information and Computation*, 206(12):1383–1393, 2008. 23

[Fra87] P. Frankl. Cops and robbers in graphs with large girth and cayley graphs. *Discrete Applied Mathematics*, 17(3):301–305, 1987. 102, 103

[FT03] F.V. Fomin and D.M. Thilikos. On the monotonicity of games generated by symmetric submodular functions. *Discrete Applied Mathematics*, 131(2):323–335, 2003. 43

[FT06] F. Fomin and D. Thilikos. Dominating sets in planar graphs: Branch-width and exponential speed-up. *SIAM Journal on Computing*, 36(2):281–309, 2006. 17

[Gal67] T. Gallai. Transitiv orientierbare graphen. *Acta Mathematica Academiae Scientiarum Hungarica*, 18(1-2):25–66, 1967. 2

[Gan12] R. Ganian. Using neighborhood diversity to solve hard problems. *CoRR*, abs/1201.3091, 2012. 172

[GHH05] W. Goddard, S.M. Hedetniemi, and S.T. Hedetniemi. Eternal security in graphs. *Journal of Combinatorial Mathematics and Combinatorial Computing*, 52:169–180, 2005. 106

[GHK+10] R. Ganian, P. Hliněný, J. Kneis, D. Meister, J. Obdržálek, P. Rossmanith, and S. Sikdar. Are there any good digraph width measures? In *Parameterized and Exact Computation*, volume 6478 of *Lecture Notes in Computer Science*, pages 135–146. Springer Berlin Heidelberg, 2010. 5, 23

[Gim03] J. Gimbel. Some remarks on the convexity number of a graph. *Graphs and Combinatorics*, 19(3):357–361, 2003. 148

[GK08] J.L. Goldwasser and W.F. Klostermeyer. Tight bounds for eternal dominating sets in graphs. *Discrete Mathematics*, 308(12):2589–2593, 2008. 106

[GMN+13] F. Giroire, D. Mazauric, N. Nisse, S. Pérennes, and R. Soares. Connected surveillance game. In *Sirocco*, 2013. 7

[GR95] A.S. Goldstein and E.M. Reingold. The complexity of pursuit on a graph. *Theoretical Computer Science*, 143(1):93–112, 1995. 4, 103

[HK08] P. Hunter and S. Kreutzer. Digraph measures: Kelly decompositions, games, and orderings. *Theoretical Computer Science*, 399(3):206–219, 2008. 23, 26

[HLT93] F. Harary, E. Loukakis, and C. Tsouros. The geodetic number of a graph. *Mathematical and Computer Modelling*, 17(11):89–95, 1993. 147

[HM06] G. Hahn and G. MacGillivray. A note on k-cop, l-robber games on graphs. *Discrete Mathematics*, 306(19–20):2492–2497, 2006. 104

[Hun06] Paul Hunter. Losing the +1 or directed path-width games are monotone, 2006. unpublished result. 27

[JP89] M.S. Jacobson and K. Peters. Complexity questions for n-domination and related parameters. *Congressus Numerantium*, 68:7–22, 1989. 149

[JRST01] T. Johnson, N. Robertson, P.D. Seymour, and R. Thomas. Directed tree-width. *Journal of Combinatorial Theory, Series B*, 82(1):138–154, 2001. 23, 24

[JS03] N. Jose and A.K. Somani. Connection rerouting/network reconfiguration. In *Design of Reliable Communication Networks, 2003. (DRCN 2003). Proceedings. Fourth International Workshop on*, pages 23–30, 2003. 29, 30

[KC85] L.M. Kirousis and Papadimitriou C.H. Interval graphs and searching. *Discrete Mathematics*, 55(2):181–184, 1985. 4

[Kim04] B.K. Kim. A lower bound for the convexity number of some graphs. *Journal of Applied Mathematics and Computing*, 14(1-2):185–191, 2004. 150

[Kin92] N.G. Kinnersley. The vertex separation number of a graph equals its path-width. *Information Processing Letters*, 42(6):345–350, 1992. 20

[KLM97] T.M. Kroeger, D.D.E. Long, and J.C. Mogul. Exploring the bounds of web latency reduction from caching and prefetching. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems on USENIX Symposium on Internet Technologies and Systems*, USITS'97, pages 2–2, 1997. 111

[KLNS12] A. Kosowski, B. Li, N. Nisse, and K. Suchan. k-chordal graphs: From cops and robber to compact routing via treewidth. In *Automata, Languages, and Programming*, volume 7392 of *Lecture Notes in Computer Science*, pages 610–622. Springer Berlin Heidelberg, 2012. 4, 104

[Klo94] T. Kloks. *Treewidth: Computations and Approximations*. Lecture Notes in Computer Science. Springer, 1994. 15

[Klo07] O. Kloster. A solution to the angel problem. *Theoretical Computer Science*, 389(1–2):152–161, 2007. 107

[KM09] W.F. Klostermeyer and C.M. Mynhardt. Edge protection in graphs. *The Australasian Journal of Combinatorics*, 45:235–250, 2009. 106

[KM11] W.F. Klostermeyer and C.M. Mynhardt. Graphs with equal eternal vertex cover and eternal domination numbers. *Discrete Mathematics*, 311(14):1371–1379, 2011. 106

[KN13] M.M. Kanté and L. Nourine. Polynomial time algorithms for computing a minimum hull set in distance-hereditary and chordal graphs. In *SOFSEM 2013: Theory and Practice of Computer Science*, volume 7741 of *Lecture Notes in Computer Science*, pages 268–279. Springer Berlin Heidelberg, 2013. 5, 6, 148

[KO08] S. Kreutzer and S. Ordyniak. Digraph decompositions and monotonicity in digraph searching. In *Graph-Theoretic Concepts in Computer Science*, volume 5344 of *Lecture Notes in Computer Science*, pages 336–347. Springer Berlin Heidelberg, 2008. 4, 25

[KP86] L.M. Kirousis and C.H. Papadimitriou. Searching and pebbling. *Theoretical Computer Science*, 47(0):205–218, 1986. 4, 20

[Kur30] K. Kuratowski. Sur le probleme des courbes gauches en topologie. *Fundamenta Mathematicae*, 15:271–283, 1930. 3

[Kut04] M. Kutz. *The Angel Problem, Positional Games, and Digraph Roots*. PhD thesis, Freie Universität Berlin, 2004. 107

[Kut05] M. Kutz. Conway's angel in three dimensions. *Theoretical Computer Science*, 349(3):443–451, 2005. 107

[Lam12] M. Lampis. Algorithmic meta-theorems for restrictions of treewidth. *Algorithmica*, 64:19–37, 2012. 172, 173

[LaP93] A.S. LaPaugh. Recontamination does not help to search a graph. *Journal of the ACM*, 40(2):224–245, 1993. 22

[LP12] L. Lu and X. Peng. On meyniel's conjecture of the cop number. *Journal of Graph Theory*, 71(2):192–205, 2012. 103

[Mö90] R.H. Möhring. Graph problems related to gate matrix layout and pla folding. In *Computational Graph Theory*, volume 7 of *Computing Supplementum*, pages 17–51. Springer Vienna, 1990. 20

[Mö96] R.H. Möhring. Triangulating graphs without asteroidal triples. *Discrete Applied Mathematics*, 64(3):281–287, 1996. 15

[Má07] A. Máthé. The angel of power 2 wins. *Combinatorics, Probability and Computing*, 16(3):363–374, 2007. 107

[Mam13] M. Mamino. On the computational complexity of a game of cops and robbers. *Theoretical Computer Science*, 477(0):48–56, 2013. 4, 103

[MHG+81] N. Megiddo, S.L. Hakimi, M.R. Garey, D.S. Johnson, and C.H. Papadimitriou. The complexity of searching a graph. In *Foundations of Computer Science, 1981. SFCS '81. 22nd Annual Symposium on*, pages 376–385, 1981. 4

[Mog96] J.C. Mogul. Hinted caching in the web. In *Proceedings of the 7th workshop on ACM SIGOPS European workshop: Systems support for worldwide applications*, EW 7, pages 103–108, 1996. 111

[MS88] B. Monien and I.H. Sudborough. Min cut is np-complete for edge weighted trees. *Theoretical Computer Science*, 58(1–3):209–229, 1988. 18

[MS89] F. Makedon and I.H. Sudborough. On minimizing width in linear layouts. *Discrete Applied Mathematics*, 23(3):243–265, 1989. 22

[NS13] N. Nisse and R. Soares. On the monotonicity of process number. In *LAGOS*, 2013. 6

[NW83] R. Nowakowski and P. Winkler. Vertex-to-vertex pursuit in a graph. *Discrete Mathematics*, 43(2–3):235–239, 1983. 101, 102, 105

[OMK+79] T. Ohtsuki, H. Mori, E.S. Kuh, T. Kashiwabara, and T. Fujisawa. One-dimensional logic gate assignment and interval graphs. *Circuits and Systems, IEEE Transactions on*, 26(9):675–684, 1979. 14

[Par78] T.D. Parsons. Pursuit-evasion in a graph. In *Theory and Applications of Graphs*, volume 642 of *Lecture Notes in Mathematics*, pages 426–441. Springer Berlin Heidelberg, 1978. 4, 21

[PM96] V.N. Padmanabhan and J.C. Mogul. Using predictive prefetching to improve world wide web latency. *SIGCOMM Computer Communication Review*, 26(3):22–36, 1996. 111

[Qui83] A. Quilliot. *Problèmes de jeux, de point Fixe, de connectivité de represésentation sur des graphes, des ensembles ordonnés et des hypergraphes.* PhD thesis, Thèse d'Etat, Université de Paris IV, 1983. 101

[Qui86] A. Quilliot. Some results about pursuit games on metric spaces obtained through graph theory techniques. *European Journal of Combinatorics*, 7(1):55–66, 1986. 104

[RS83] N. Robertson and P.D. Seymour. Graph minors. i. excluding a forest. *Journal of Combinatorial Theory, Series B*, 35(1):39–61, 1983. 13, 15, 43

[RS84] N. Robertson and P.D. Seymour. Graph minors. iii. planar tree-width. *Journal of Combinatorial Theory, Series B*, 36(1):49–64, 1984. 2

[RS91] N. Robertson and P.D. Seymour. Graph minors. x. obstructions to tree-decomposition. *Journal of Combinatorial Theory, Series B*, 52(2):153–190, 1991. 19, 43

[RS95] N. Robertson and P.D. Seymour. Graph minors .xiii. the disjoint paths problem. *Journal of Combinatorial Theory, Series B*, 63(1):65–110, 1995. 2

[RS04] N. Robertson and P. Seymour. Graph minors. xx. wagner's conjecture. *Journal of Combinatorial Theory, Series B*, 92(2):325–357, 2004. 3, 13, 15

[SB91] P.D. Seymour and D. Bienstok. Monotonicity in graph searching. *Journal of Algorithms*, 12:239–245, 1991. 4, 22

[Sch98] A. Schrijver. *Theory of Linear and Integer Programming*. Wiley Series in Discrete Mathematics & Optimization. John Wiley & Sons, 1998. 131

[Sch01] B.S.W. Schröder. The copnumber of a graph is bounded by [3/2 genus(g)] + 3. In *Categorical Perspectives*, pages 243–263. Birkhäuser Boston, 2001. 104

[Soa13] R. Soares. Fractional combinatorial games on graphs. In *15èmes Rencontres Francophones sur les Aspects Algorithmiques des Télécommunications (AlgoTel)*, 2013. 7

[Sol09] F. Solano. Analyzing two conflicting objectives of the wdm lightpath reconfiguration problem. In *Proceedings of the Global Communications Conference (GLOBECOM)*, pages 1–7, 2009. 31

[SP09] F. Solano and M. Pióro. A mixed-integer programing formulation for the lightpath reconfiguration problem. In *VIII Workshop on G/MPLS Networks (WGN8)*, 2009. 31

[SS11] A. Scott and B. Sudakov. A bound for the cops and robbers problem. *SIAM Journal on Discrete Mathematics*, 25(3):1438–1442, 2011. 103

[SS13] R.M. SAMPAIO and J.L. Szwarcfiter. The convexity of induced paths of order three. In *Proceedings of the Latin-American Algorithms, Graphs and Optimization Symposium (LAGOS)*, pages 196–202, 2013. 149

[ST93] P.D. Seymour and R. Thomas. Graph searching and a min-max theorem for tree-width. *Journal of Combinatorial Theory, Series B*, 58(1):22–33, 1993. 3, 4, 19, 20, 32, 41, 43

[ST94] P.D. Seymour and R. Thomas. Call routing and the ratcatcher. *Combinatorica*, 14(2):217–241, 1994. 2, 3, 17, 18, 48

[Tar85] R.E. Tarjan. Decomposition by clique separators. *Discrete Mathematics*, 55(2):221–232, 1985. 148

[TCH08] M. Tedder, D. Corneil, and C. Habib, M.and Paul. Simpler linear-time modular decomposition via recursive factorizing permutations. In *Automata, Languages and Programming*, volume 5125 of *Lecture Notes in Computer Science*, pages 634–645. Springer Berlin Heidelberg, 2008. 3

[The08] D.O. Theis. The cops & robber game on series-parallel graphs. arXiv:0712.2908v2, 2008. 104

[Thi00] D.M. Thilikos. Algorithms and obstructions for linear-width and related search parameters. *Discrete Applied Mathematics*, 105(1–3):239–271, 2000. 3, 4, 17, 19, 43, 48

[TUK95] A. Takahashi, S. Ueno, and Y. Kajitani. Mixed searching and proper-path-width. *Theoretical Computer Science*, 137(2):253–268, 1995. 22

[Vaz04] V.V. Vazirani. *Approximation Algorithms*. Springer, 2004. 138

[WLZC12] Z. Wang, F.X. Lin, L. Zhong, and M. Chishtie. How far can client-only solutions go for mobile browser speed? In *Proceedings of the 21st international conference on World Wide Web*, WWW '12, pages 31–40, 2012. 111

[YC07] B. Yang and Y. Cao. Directed searching digraphs: Monotonicity and complexity. In *Theory and Applications of Models of Computation*, volume 4484, pages 136–147. Springer Berlin Heidelberg, 2007. 4

[YDA09] B. Yang, D. Dyer, and B. Alspach. Sweeping graphs with large clique number. *Discrete Mathematics*, 309(18):5770–5780, 2009. 23