COMPLEXITY AND ALGORITHMS RELATED TO TWO CLASSES OF GRAPH PROBLEMS

Carlos Vinícius Gomes Costa Lima

Tese de Doutorado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Doutor em Engenharia de Sistemas e Computação.

Orientadores: Jayme Luiz Szwarcfiter
Mitre Costa Dourado
Uéverton dos Santos Souza

Rio de Janeiro
Abril de 2017

COMPLEXITY AND ALGORITHMS RELATED TO TWO CLASSES OF GRAPH PROBLEMS

Carlos Vinícius Gomes Costa Lima

TESE SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE DOUTOR EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Examinada por:

RIO DE JANEIRO, RJ – BRASIL
ABRIL DE 2017

Lima, Carlos Vinícius Gomes Costa

Complexity and Algorithms Related to Two Classes of Graph Problems/Carlos Vinícius Gomes Costa Lima. – Rio de Janeiro: UFRJ/COPPE, 2017.

XI, 227 p.: il.; 29, 7cm.

Orientadores: Jayme Luiz Szwarcfiter
Mitre Costa Dourado
Uéverton dos Santos Souza

Tese (doutorado) – UFRJ/COPPE/Programa de Engenharia de Sistemas e Computação, 2017.

Bibliography: p. 113 – 124.

1. Reversible Processes. 2. Convexity. 3. Threshold Processes. 4. AND/OR Model. 5. Graph Editting. I. Szwarcfiter, Jayme Luiz *et al.* II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

"Lasciate ogne speranza, voi ch'entrate."
"Abandon all hope, ye who enter here."
(Hell's Gate)

— D. Alighieri, *Divine Comedy*

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Doutor em Ciências (D.Sc.)

COMPLEXIDADE E ALGORITMOS RELACIONADOS A DUAS CLASSES DE PROBLEMAS DE GRAFOS

Carlos Vinícius Gomes Costa Lima

Abril/2017

Orientadores: Jayme Luiz Szwarcfiter
Mitre Costa Dourado
Uéverton dos Santos Souza

Programa: Engenharia de Sistemas e Computação

Esta tese aborda problemas associados a conversões em grafos e de edição pela remoção de um emparelhamento. Estudamos processos f-reversíveis, que são aqueles associados a um valor de limiar para cada vértice e cuja dinâmica depende da quantidade de vizinhos com estado contrário para cada vértice. Estabelecemos um limite superior justo para o tamanho do período e transiente, caracterizamos todas as árvores que alcançam o transiente máximo em processos 2-reversíveis e mostramos que determinar o tamanho de um conjunto conversor mínimo é NP-difícil.

Mostramos que o modelo AND-OR define uma convexidade sobre grafos. Mostramos resultados de NP-completude e algoritmos eficientes para certos parâmetros de convexidade para esta nova, assim como algoritmos aproximativos.

Introduzimos o conceito de processos de limiar generalizados, onde mostramos resultados de NP-completude e algoritmos eficientes para ambas as versões não relaxada e relaxada.

Estudamos o problema de decidir se um dado grafo admite uma remoção de um emparelhamento de modo a remover todos os ciclos. Mostramos que este problema é NP-difícil mesmo para grafos subcúbicos, mas admite solução eficiente para várias classes de grafos.

Estudamos o problema de decidir se um dado grafo admite uma remoção de um emparelhamento de modo a remover todos os ciclos ímpares. Mostramos que este problema é NP-difícil mesmo para grafos planares com grau limitado, mas admite solução eficiente para algumas classes de grafos. Mostramos também resultados parametrizados.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Doctor of Science (D.Sc.)

COMPLEXITY AND ALGORITHMS RELATED TO TWO CLASSES OF GRAPH PROBLEMS

Carlos Vinícius Gomes Costa Lima

April/2017

Advisors: Jayme Luiz Szwarcfiter
Mitre Costa Dourado
Uéverton dos Santos Souza

Department: Systems Engineering and Computer Science

This thesis addresses the problems associated with conversions on graphs and editing by removing a matching. We study the f-reversible processes, which are those associated with a threshold value for each vertex, and whose dynamics depends on the number of neighbors with different state for each vertex. We set a tight upper bound for the period and transient lengths, characterize all trees that reach the maximum transient length for 2-reversible processes, and we show that determining the size of a minimum conversion set is NP-hard.

We show that the AND-OR model defines a convexity on graphs. We show results of NP-completeness and efficient algorithms for certain convexity parameters for this new one, as well as approximate algorithms.

We introduce the concept of generalized threshold processes, where the results are NP-completeness and efficient algorithms for both non relaxed and relaxed versions.

We study the problem of deciding whether a given graph admits a removal of a matching in order to destroy all cycles. We show that this problem is NP-hard even for subcubic graphs, but admits efficient solution for several graph classes.

We study the problem of deciding whether a given graph admits a removal of a matching in order to destroy all odd cycles. We show that this problem is NP-hard even for planar graphs with bounded degree, but admits efficient solution for some graph classes. We also show parameterized results.

Contents

List of Figures **ix**

List of Tables **xi**

1 Introduction **1**

1.1 A Short Overview of the Problems 2

1.1.1 Discrete Conversion Processes on Graphs 2

1.1.2 Editing Problems 5

1.2 Preliminaries 6

1.2.1 Basic Definitions 6

1.2.2 Graph Convexity 9

2 f-Reversible Processes **12**

2.1 Dissemination on Graphs 12

2.2 Reversible Processes 14

2.2.1 f-Reversible Processes 14

2.2.2 Formalization 15

2.2.3 Threshold Networks 17

2.2.4 The Periodic Behavior 18

2.2.5 The Potential Function 23

2.2.6 The Minimum f-Conversion Set Problem 25

2.3 Results 30

2.3.1 A Tight Upper Bound on the Transient Length of f-Reversible Processes 30

2.3.2 The Transient Length of 2-Reversible Processes on Trees 36

2.3.3 NP-Completeness of f-Conversion Set 42

3 Some Irreversible Processes with Thresholds **47**

3.1 Threshold Process Based on the AND-OR Model 48

3.1.1 The AND-OR Model 48

3.1.2 The And/Or-Convexity 50

3.2 Computing the Convexity Parameters . . . 51
3.2.1 The Convexity Number . . . 52
3.2.2 The Interval Number . . . 53
3.2.3 The Hull Number . . . 59
3.2.4 The Carathéodory Number . . . 63
3.3 Generalized Threshold Processes on Graphs . . . 64
3.3.1 Hardness results . . . 66
3.3.2 Efficient algorithms for trees . . . 69

4 Editing Problems by the Removal of a Matching **77**
4.1 Obtaining a Forest . . . 78
4.2 Obtaining a Bipartite Graph . . . 84
4.2.1 Preliminaries . . . 84
4.2.2 A linear Time Algorithm for Subcubic Graphs . . . 86
4.2.3 NP-Completeness for **Odd Decycling Matching** . . . 88
4.3 Polynomial Time Results . . . 99
4.3.1 Fixed-Parameter Tractability . . . 105

5 Conclusion **109**
5.1 Open Problems . . . 110
5.1.1 f-Reversible Processes . . . 110
5.1.2 AND/OR-Convexity . . . 110
5.1.3 Generalized Threshold Process . . . 111
5.1.4 Decycling with a Matching . . . 111
5.1.5 Odd Decycling with a Matching . . . 111

Bibliography **113**

Appendices **125**
Appendix A: On f-Reversible Processes on Graphs . . . 125
Appendix B: A Computational Study of f-Reversible Processes on Graphs 132
Appendix C: And/Or-Convexity: A Graph Convexity Based on Processes and Deadlock Models . . . 162
Appendix D: Generalized Threshold Processes on Graphs . . . 182
Appendix E: Decycling with a Matching . . . 196
Appendix F: Decycling with a Matching . . . 198
Appendix G: Odd Decycling with a matching . . . 206

List of Figures

1.1 Example of an initial vertex set in a P_3 convexity. 10

2.1 Example of the dynamic of an f-reversible process. 19
2.2 Phase graph of the process presented in Figure 2.1. 19
2.3 Phase graphs of the f-reversible process defined in Figure 2.1. 20
2.4 Example of an f-reversible process with unbounded transient length. 21
2.5 Phase graph of the process described in Figure 2.4. 21
2.6 Example of threshold network with period greater than 2. 22
2.7 Configurations of an f-reversible process over P_5. 25
2.8 The initial configuration of a 2-reversible process on a tree whose gray vertices have state 1. 25
2.9 Time evolution of the potential (dotted line) and energy (filled line) functions. 26
2.10 Example of a non trivial f-conversion set. 26
2.11 Example of a superset that is not an f-conversion set of G. 27
2.12 Example of configuration satisfying Observation 2.6. 28
2.13 Transient phase of $R_f(C_7, c_0)$. 36
2.14 All non-periodic configurations with three and four vertices. 37
2.15 Representation of configuration c_{t_2} in Sub-case 1.1. 37
2.16 Representation of tree T' obtained from a tree T, such that $\tau_2(T') \geq \tau_2(T)$. 38
2.17 All configurations with $\tau(c_0) \geq n-3$, such that $X_1 = Y_1 = \emptyset$, $d(v) = 2$ and $n \geq 5$. 39
2.18 Representation of the initial configurations of trees in which $\tau_2(c_0) = n-3$ and all non-leaf vertices reach their final states exactly one time step after all of their children. 39
2.19 Representation of tree T, where $v \in S_1(t)$, for all $t \leq \tau(c_0) - 1$. 40
2.20 Representation of a general initial configuration of trees in which $\tau(c_0) = n-3$ and at least one non-leaf vertex reaches its final state after all of its children. 41

2.21 Representation of trees T with $\tau(c_0) = n - 3$, in which $y = 0$ and n even. 41

2.22 A labelling with $r_f(G)$ vertices satisfying $F = (X_1 \vee \overline{X_2}) \wedge (\overline{X_1} \vee X_2 \vee X_3) \wedge (X_2 \vee \overline{X_3} \vee X_4) \wedge (\overline{X_3} \vee \overline{X_4})$. 45

3.1 Example of transformation from graph G to G' in Theorem 3.5. The black vertices are a minimum vertex cover of G in Figure 3.1a and a minimum interval set of G' in Figure 3.1b. 54

3.2 Representation of a hull set of G given by the gray sets. 61

3.3 Example of a graph and τ function. 65

4.1 Some examples of forbidden subgraphs. 85

4.2 The associated graph $G_{F'}$ obtained from $F = (x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_3 \vee x_4) \wedge (x_1 \vee x_4 \vee x_5) \wedge (x_2 \vee x_3 \vee x_5)$. The black vertices correspond to clauses. 90

4.3 The head graph and its odd decycling matching M, which is represented by the stressed edges. Vertices with same color belong to the same part in the bipartition of $G - M$. 91

4.4 Variable gadget G_{c_j} in Theorem 4.15. 93

4.5 Clause gadget G_{c_j} in Theorem 4.15. 93

4.6 Clause gadget G_{c_j} in Theorem 4.17. 95

4.7 Variable gadget G_{x_i} in Theorem 4.17. Each pair of edges with one no endvertex connects G_{x_i} to one clause gadget G_{c_a}, G_{c_b}, or G_{c_c}, where $x_i \in (c_a \cap c_b \cap c_c)$. 95

4.8 All possible configurations given by the removal of an odd decycling matching M, represented by the stressed edges, from a clause gadget G_{c_j} of three literals. The colors in the vertices represent the bipartition of $G_{c_j} - M$. 97

4.9 All possible configurations given by the removal of an odd decycling matching M, represented by the stressed edges, from a variable gadget G_{x_i}. The colors in the vertices represent the bipartition of $G_{x_i} - M$. 98

4.10 The modified variable gadget. 99

4.11 All possible configurations given by the removal of an odd decycling matching M from the modified variable gadget. 99

4.12 All possible neighborhoods of v of a claw-free graph $G \in \mathcal{BM}$. 100

List of Tables

1.1 Terms and basic symbols of graph theory used in this thesis. 9

2.1 Sets $S_1(t)$, $S_2(t)$, values $op_t(v_i)$, $1 \leq i \leq 5$, and potential function $P(t)$. 25
2.2 Resume of the main exact results on $r_f(G)$. 29
2.3 Sets $S_1(t)$, $S_2(t)$, values $op_{t+1}(v_i)$, $1 \leq i \leq 5$, and potential function $P'(t)$. 31

Chapter 1

Introduction

"And a mind needs books like a sword needs a whetstone."
(Tyrion Lannister)

— G. Martin, *A Game of Thrones*

Graph theory is a branch of the math that is used by the computer science to describe and model several real and theoretical problems. In this thesis we concern on some different problems regarding mainly distributed computing and graph editing problems, where its structure is modified in order to obtain some property. In particular, we study a specific reversible dynamical process on graphs that we call as *f-reversible process*, where each vertex has a *state* and an integer label. The next problem consists of also labeling the vertices of a given graph G into two types (AND- and OR-vertices). This labeling is based on the AND-OR model [14] that describes *deadlocks* in a distributed computation. The third problem is a generalization of *threshold processes* on graphs, that are processes that start with a subset of vertices S, and a new vertex v is included in S whenever some condition on the neighborhood of v is satisfied. Finally we study two *graph editing* problems based on the removal of a subset of edges that induces a matching, in order to know whether the obtained graph is a tree, as well as if it is a bipartite graph, respectively.

The results obtained in this thesis can be found at the final of each chapter in the appendixes, as well as all of the complete proofs. In the text we have chosen to hide some proofs for simplicity. Next we provide a short overview of the problems studied. More details on these problems can be founded in the next three chapters.

1.1 A Short Overview of the Problems

1.1.1 Discrete Conversion Processes on Graphs

A discrete conversion process on a graph $G = (V, E)$ is one that, starting with an initial subset $S \subseteq V(G)$, and following a determined rule at each discrete time step $t \geq 0$, the set S is eventually modified. If vertices can only be added to S and S cannot be modified, then the process is called *irreversible*. Otherwise, if the process follows in such a way that some vertices of S can be removed from the current S as well as vertices from $V(G) \setminus S$ can be added to the current S, then the process is said to be *reversible*.

In this thesis we deal with discrete conversion processes based on some restricted rules that are employed in a wide number of branches of the computer science, specially on the distributed computing. Several parameters studied here on these processes are based on *graph convexities*.

The convexity concept comes from the Euclidean and Arquimedes works on geometry. A *convex set* in their works is a set S such that every line segment between two points of S remains in S. The *convex hull* of a set V of points is the set of all points obtained by the convex combination of the points in V. We can model a convexity on graphs considering for example the set of points as a set S of vertices, and some vertices are added to S according to some condition, that can be based on the distances between those points, as in the *geodetic convexity*, as well as some neighborhood condition, as in the P_3 *convexity*.

Here we consider discrete conversion processes on a graph G such that the conversion rules depend on the neighborhood of the vertices. An example can be done by the P_3 convexity, where given an initial subset S of the vertices, a vertex $v \in V(G) \setminus S$ is added to S if and only if v has at least two neighbors in S. Moreover this graph convexity is an example of an irreversible discrete conversion process. Since we consider only finite graphs, at some time this process ends, that is, the set S is such that no vertices can be added to it. Natural parameters arise from such processes, as the size of the final set S related to the initial one, the number of time steps required to reach this final S, the maximum number of new vertices it can be reached in just one time step, the minimum number of vertices required to form S in order that the entire vertex set is converted, and so on. Some graph convexity parameters will be presented in the final of this chapter.

For reversible processes, another natural parameter can be proposed. Since the graph is finite, and if only a finite number of "states" for each vertex (for example if a vertex v is in S or not) is considered, and the conversion rule is deterministic, then the process finishes in a periodic configuration, that is, a state assignment to the vertices that can be reached again after a finite number p of time steps. Such a parameter p is the size of the periodic phase of the reversible process. Next we present an overview of the considered discrete conversion processes.

f-Reversible Processes

The main motivation for the study of the f-reversible processes comes from distributed computing, and some graph convexity parameters. It consists on an assignment to each vertex of G of a non-negative integer $f(v)$, as well as an initial state between two, say 0 and 1, and represented by $c_0(v)$. The process follows in such a way that each vertex changes its state if and only if at least $f(v)$ of its neighbors have the opposite state. The changes are done synchronously, that is, all vertices do the test in its neighborhood at the same time. Moreover, the process takes a non bounded number of time steps.

Such processes are classified as reversible, since each vertex can change its state from 0 to 1, as well as from 1 to 0. They represent several real problems in many different branches of the science, such as astrophysics [121], physical and biological cell population simulation [95, 136], percolation [9], marketing strategies [56, 64, 92], neural networks [81], local interaction games [106, 109] and in distributed computing [71, 72, 87, 98, 111, 120], manly in the study of consensus and distributed voting in a network.

We study the periodic behavior of f-reversible processes, since such processes acquire a periodic phase after a finite number of time steps. Moreover, we consider the complexity of determining the minimum number of vertices with the same initial state 1 required to obtain an f-reversible process where all vertices have state 1 in the periodic phase. Such a parameter is similar to the *hull number* in graph convexities.

AND/OR-Convexity

The next problem studied is also based on distributed computing, where we relate some classical convexity parameters to blocking sets that cause *deadlocks*. Let V denote a set of processes in a distributed computation. Informally, as described by Barbosa and Benevides [14], a *deadlock* is said to exist in this computation if a subset $S \subseteq V$ can be identified whose members are all blocked due to the occurrence of some condition that can only be relieved by members of the same subset S. Such a set S is called a *blocking set*.

We study the AND-OR model, where each vertex receives a label AND or a label OR, where an AND-vertex (resp., OR-vertex) depends on the computation of all (resp., at least one) of its neighbors. We define a graph convexity, that we call *AND/OR-convexity* based on this model, such that a set $S \subseteq V(G)$ is *convex* if and only if every AND-vertex (resp., OR-vertex) $v \in V(G) \setminus S$ has at least one (resp., all) of its neighbors in $V(G) \setminus S$.

We relate some classical convexity parameters, such as the *convexity number, hull number, interval number*, and *Charathéodory number* to blocking sets that cause deadlock situations in the AND-OR model. In particular, we show that those parameters in the AND/OR-convexity represent for example the sizes of minimum or maximum blocking sets, and also the computation time until system stability is reached. Finally, a study on the complexity of computing such parameters in the AND/OR-convexity is provided.

Generalized Threshold Processes on Graphs

In this problem we consider a process on a graph $G = (v, E)$ starting with some given subset of vertices S, and such that every vertex v has a list $\tau(v)$ of subsets of its neighbors. Such a process is irreversible and iteratively adds to S all vertices u of G outside of S for which the intersection of the current set S with the neighborhood of u in G belongs to a $\tau(u)$. Based on discrete convexity notions, we study the corresponding interval number, called *τ-interval number*, where only one iteration is executed, and present some results on the hull number, called *τ-hull number*, where the number of iterations is unbounded. Special choices of the function τ allow to include several well studied graph processes and parameters within this framework. We call such processes as *τ-threshold processes.*

A motivation for the study of τ-threshold processes on graphs is that they generalize some gates of logical circuits. For example, an `and`-gate v outputs true only if all of its inputs are true; on the other hand, for a `xor`-gate outputs true it must receive an odd number of positive inputs. A vertex with $\tau(v)$ equals its neighborhood operates according to an `and`-gate, and when $\tau(v) = \{N \in 2^{N_G(u)} : |N| \mod 2 = 1\}$ the vertex operates as an `xor`-gate, where $V_G(u)$ denotes the neighborhood of u in a graph G.

We also consider a relaxed version of the above notions, where a vertex u from $V(G) \setminus S$ belongs to the interval of S if $N_G(u) \cap S$ *contains* some set from $\tau(u)$ instead of requiring to be equal to some such a set. This process is called *relaxed τ-threshold process*, and the analogous parameters such as the *relaxed τ-interval number* and the *relaxed τ-hull number*

Special choices for τ lead to many well known graph parameters. If $\tau(u) = 2^{N_G(u)} \setminus \{\emptyset\}$ for every vertex u of some graph G, then the τ-interval number coincides with the *domination number* $\gamma(G)$ [88]. Alternatively, if $\tau(u) = \binom{N_G(u)}{1}$, then the relaxed τ-interval number also coincides with the domination number $\gamma(G)$. More generally, if $\tau(u) = \binom{N_G(u)}{k}$ for some positive integer k, then the relaxed τ-interval number coincides with the *k-domination number* $\gamma_k(G)$ [34, 43, 69, 88, 126, 142], and the relaxed τ-threshold processes correspond to the processes considered in [2, 34, 39, 141].

1.1.2 Editing Problems

Given a graph $G = (V, E)$ and a graph property Π, the Π *edge-deletion problem* consists in determining the minimum number of edges required to be removed in order to obtain a graph satisfying Π [28]. Given an integer $k \geq 0$, the Π *edge-deletion decision problem* asks for a set $F \subseteq E(G)$ with $|F| \leq k$, such that the obtained graph by the removal of F satisfies Π. Both versions have received widely attention on the study of their complexity, where we can cite [5, 28, 76, 86, 113, 138, 139] and references therein for applications.

Here we investigate two variants of this kind of graph problem, where the required set of edges to be removed induces a matching.

Decycling with a Matching

Destroying all cycles of a given graph by removing vertices or edges is a classical theme. Clearly, the minimum number of edges of a connected graph of order n and size m whose removal destroys all cycles is exactly $m - n + 1$, and standard minimum spanning tree algorithms allow to solve even weighted optimization versions. Contrary to this, the minimum number of vertices whose removal destroys all cycles (or produces a tree) is a difficult parameter [17, 19, 66, 80, 125].

We consider a specific graph editing problem, where we ask whether a given graph G admits a matching whose removal destroys all cycles of the graph. In other words, we want to know if G is the union of a forest and a matching. We proof that such a decision problem is NP-complete even for subcubic graphs, and we show a set of polynomial time algorithms for some classical graph classes.

Odd Decycling with a matching

When the obtained graph is required to be bipartite, the corresponding edge-(vertex-) deletion problem is called *edge (vertex) bipartization* [1, 38, 74] or *edge (vertex) frustration* [140]. Choi, Nakajima, and Rim [38] showed that the edge bipartization decision problem is NP-complete even for cubic graphs.

Furmańczyk, Kubale, and Radziszowski [74] considered vertex bipartization of cubic graphs by the removal of an independent set. We study the analogous edge deletion decision problem, that is, the problem of determining whether a finite, simple, and undirected graph G admits a removal of a set of edges that is a matching in G in order to obtain a bipartite graph.

This problem is also equivalent to the problem of determining whether a graph G admits an $(1, 1)$-coloring, which is a 2-coloring of $V(G)$ in which each color class induces a graph of maximum degree at most 1. We show that such a decision problem is NP-complete even for planar graphs of maximum degree 4, but can be solved in linear time in graphs of maximum degree 3. We also present polynomial time algorithms for (claw, paw)-free graphs, graphs containing only triangles as odd cycles, graphs with bounded dominating sets, and P_5-free graphs. In addition, we show that the problem is fixed-parameter tractable when parameterized by clique-width, which implies polynomial time solvability for many interesting graph classes such as distance-hereditary graphs and outerplanar graphs. Finally a $2^{vc(G)}.n$ algorithm, and a kernel having at most $2.nd(G)$ vertices are presented, where n is the order of G, and $vc(G)$ and $nd(G)$ are the vertex cover number and the neighborhood diversity of the input graph, respectively.

1.2 Preliminaries

In this section, we provide the definitions concerning graphs that will be used in the text, and some others. In order to take more details on the definitions on graph theory we refer to the book of Bondy and Murty [22]. We also present definitions concerning graph convexity in the final of this section.

In this thesis we do not address in detail the questions involving complexity classes, although we use the usual notions that an algorithm whose execution time is limited by a polynomial in the input size is efficient, and that a problem in the class of problems NP-difficult is probably intractable. Thus, whenever we consider computational aspects of the problems involved, we will be looking for an efficient algorithm of smallest complexity as possible, or an NP-completeness proof for a simplest instance as possible.

1.2.1 Basic Definitions

For some function $f : A \to B$ and $S \subseteq A$, we denote by $\text{Im}_f = \{f(x) \in B : x \in A\}$, and by $\text{Im}_f(S) = \{f(x) \in B : x \in S\}$. We call Im_f as *image* of f. Moreover, we denote by $\mathbb{N}$ the set of non-negative integer numbers, that is, $\mathbb{N} = \{0, 1, 2, \dots\}$.

We denote by $\tau(u) = \binom{V}{k}$ the set of all subsets of V size k, and denote the set of all subsets of a given set S by 2^S. A *family* of V is a set of subsets of V.

We say that the set of sets $V_1, V_2, \dots, V_k$ is a *partition* of a set V if and only if $V = \bigcup_{1 \leq i \leq k} V_i$, and $V_i \cap V_j = \emptyset$, for every pair $1 \leq i < j \leq k$. A *bipartition* of V occurs when $k = 2$.

A *graph* $G = (V, E)$ is an ordered pair $(V(G), E(G))$, such that V is a non empty set of elements called *vertices*, and E is a set of non ordered pairs of distinct vertices called *edges*. We denote a graph $G = (V, E)$ only by G by simplicity whenever possible. In the same way, we will denote $V(G)$ and $E(G)$ by simply V and E, respectively. We also use the notation $|V(G)| = n(G)$ (or simply n) and $|E(G)| = m(G)$ (or simply m) for the cardinalities of the vertex set and edge set of G, respectively. A graph is called *finite* if its vertex set is finite.

We say an edge e *connects* two vertices u and v and we say that u and v are the *endvertices* of e. Moreover, we say that u and v are *adjacent* if they are endvertices of a common edge, that is, they are connected by an edge. An edge e is *incident* to a vertex v if v is an endvertex of e. We denote an edge e with endvertices u and v by $e = (u, v)$.

Given disjoint subsets A and B of $V(G)$, we denote the set of all edges with one end in A and the other in B by $[A, B]$.

The set of adjacent vertices of a vertex v in a graph G is denoted by $N_G(V)$ and it is called the *neighborhood* of v. A vertex $u \in N_G(v)$ is called a *neighbor* of v in G. The *closed neighborhood* of v is the set $N_G[v] = N(v) \cup \{v\}$. The *degree* of a vertex v in G is the number of neighbors of v and it is denoted by $d_G(v)$. For a subset $S \subseteq V(G)$, let $N_G(S) = \bigcup_{v \in S} N_G(v)$ and $N_G[S] = \bigcup_{v \in S} N_G[v]$.

A *simple* graph G is a graph such that if there sexists an edge $e = (u, v)$ in G, then there is no other edge $e' = (u, v)$, that is, e is the unique edge incident to u and v. We say that a graph G is *directed* if all of its edges $e = (u, v)$ are directed from u toward v.

A graph H is a *subgraph* of a graph G if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. Given a graph G, an *induced subgraph* H of G by $S \subseteq V(G)$, denoted by $G[S]$, is such that for all $u, v \in V(H)$, if $(u, v) \in E(G)$, then $(u, v) \in E(H)$.

We denote by $G - v$ and $G - e$ the induced subgraphs of a graph G, where we eliminate a vertex v and an edge e, respectively. In the same way, we denote by $G - S$ and $G - M$ the induced subgraphs of a graph G, where we eliminate a vertex set $S \subseteq V(G)$ and an edge set $M \subseteq E(G)$, respectively.

A *path* in a graph G is a sequence $P = v_1 v_2 \ldots v_k$ of distinct vertices of G and such that $(v_i, v_{i+1}) \in E(G)$, for all $i \in \{1, \ldots, k-1\}$. A *chord* in a path is an edge connecting two non consecutive vertices of the path. An *induced path* is a path without chords. A *cycle* in a graph G is a sequence $C = v_1 v_2 \ldots v_k$, such that $P = v_1 v_2 \ldots v_{k-1}$ is a path and $v_1 = v_k$. An *induced cycle* is a cycle without chords in the path P. We denote by P_k and C_k the induced path and induced cycle by k vertices, respectively. The length of a path P_k and a cycle C_k is equals their number of edges. If there exists a path P between two distinct vertices u and v, then we say that P is a *uv-path*. The *distance* between two vertices u and v in a graph G is the size of a minimum uv-path in G. For a cycle C_k, we say that it is an *even cycle* if k is even and an *odd cycle*, otherwise.

A graph G is *connected* if there exists a path between all pair of distinct vertices of G. Otherwise, we say that G is *disconnected.* A *connected component* of G is a maximal connected subgraph of G. We denote by $w(G)$ the number of connected components of G.

A graph with no cycles is called *acyclic.* A *forest* is an acyclic graph. A *tree* is a connected acyclic graph. A tree T is *rooted* if a vertex $v \in V(T)$ is chosen as a *root* and all the other vertices are organized in *levels* $l_1, l_2, \ldots, l_h$, such that all vertices in the level l_k have distance k until v. The *height* of a rooted tree is given by the number of its levels plus one. If a vertex is in the level l_k, then its neighbor in the level l_{k-1} is its *parent* and all of its neighbors in the level l_{k+1} are its *children.*

A *chordal* graph is one that every cycle with at least 4 vertices contains a chord.

The *complement* of a graph $G = (V, E)$ is the graph $\overline{G} = (V, E')$, where $(u, v) \in E(G)$ if and only if $(u, v) \notin E'(\overline{G})$. A graph is *complete* if all of its pairs of distinct vertices are adjacent. We denote a complete graph with ℓ vertices by K_ℓ. A graph is called *empty* if its complement graph is a complete graph. A set of vertices C in a graph G is a *clique* if $G[S]$ is a complete graph. We say that $S \subseteq V(G)$ is an *independent set* if $G[S]$ is an empty graph.

A set $S \subseteq V(G)$ is a *vertex cover* of a graph G if all edges of G are incident to some vertex in S. The size of a minimum vertex cover of G is denoted by $\beta(G)$.

A set $S \subseteq V(G)$ is a *dominating set* of a graph G if all vertices of $V(G) \setminus S$ are adjacent to some vertex in S. The size of a minimum vertex cover, the *domination number* of G, is denoted by $\gamma(G)$.

A *coloring* of the vertices of a graph G is a color assignment to the vertices of G. More precisely, a coloring is a function $c : V(G) \to \mathbb{N}$. A *proper coloring* is a coloring such that adjacent vertices receive distinct colors, otherwise such a coloring is called *improper.* A graph is *k-colorable* if it admits a proper coloring with k colors, called as *k-coloring.* In the same way, a graph is *k-improper colorable* if it admits an improper coloring with k colors, called as *k-improper coloring.* The minimum number of colors required to properly color G is the *chromatic number* of G, $\chi(G)$.

A *bipartite* graph G is a graph that admits a bipartition of its vertices into sets V_1 and V_2, such that all edges of G have one endvertex in V_1 and the other one in V_2. Equivalently, there exists a partition of $V(G)$ into two sets, such that each one induces an independent set in G. A *complete bipartite* graph is a bipartite graph where each vertex from one part is adjacent to all vertices to other one. A complete bipartite graph whose parts have size r and s is denoted by $K_{r,s}$. The *claw* graph is the $K_{1,3}$ and the *paw* is the claw plus one edge.

A *regular* graph G is one that all vertices have the same degree $k > 0$, and we say that G is *k-regular.* A *cubic* graph is the 3-regular graph. A *subcubic* graph is one that all vertices have degree at most 3.

A *distance-hereditary* graph is one that all distances between any pair of vertices is the same in all connected induced subgraph.

Table 1.1 resumes the basic symbols and their meanings about graph theory. More specific definitions will be given in the next chapters as necessary.

Table 1.1: Terms and basic symbols of graph theory used in this thesis.

Symbol	Description
$G = (V, E)$	Graph G with vertex set $V(G)$ and edge set $E(G)$
$V(G)$	Set of vertices of G
$E(G)$	Set of edges of G
$n(G)$	Number of vertices of G
$m(G)$	Number of edges of G
v_i	vertex v_i
$e = (v_i, v_j)$	edge e with endvertices v_i and v_j
$d_G(v)$	Degree of v in G
$\delta(G)$	Minimum degree of G
$\Delta(G)$	Maximum degree of G
$N_G(v)$	Neighborhood of v in G
$N_G[v]$	Closed neighborhood of v in G
$G[S]$	Induced subgraph by $S \subseteq V(G)$
P_k	Path with k vertices
C_k	Cycle with k vertices
K_k	Clique or complete graph with k vertices
$K_{r,s}$	Complete bipartite graph with parts of size r and s
$\mathbb{N}$	Set of non-negative integer numbers $\mathbb{N} = \{0, 1, 2, \dots\}$
$\beta(G)$	The size of a minimum vertex cover of a graph G
$\gamma(G)$	The domination number of a graph G
$\chi(G)$	The chromatic number of a graph G
Im_f	Image of a function f
2^S	Set of all subsets of S

1.2.2 Graph Convexity

Let V be a finite set. A family $\mathcal{C}$ of V is a *convexity* on V if:

- $\mathcal{C}$ is closed under intersections;
- $\emptyset \in \mathcal{C}$;
- $V \in \mathcal{C}$.

Every $C \in \mathcal{C}$ is called a *convex set*. An example can be done by the following set $V = \{a, b, c, d, e\}$ and family $\mathcal{C} = \{\{a, b, c, d, e\}, \emptyset, \{a, b\}, \{a, c, e\}, \{b, d\}\}$.

A *graph convexity* on a graph $G = (V, E)$ is one that the convex sets represent some special property, such as *path convexities*, where if two vertices v_1 and v_2 belong to a convex set C, then all inner vertices in a minimal path between v_1 and v_2 also belong to C. We denote a graph convexity on a graph G by a pair $(G, \mathcal{C})$, where $\mathcal{C}$ is a family of convex sets.

Another graph convexities deal with the neighborhood of the vertices. For example, starting with an initial vertex set S, the P_3 convexity can be see as a dynamical process, such that every vertex $v \in V(G) \setminus S$ with two neighbors in S is added to S. The process continues until there no exist such a vertex, when S passes to be a convex set. Note that the P_3 convexity can be see as a family containing all paths of exactly three vertices. Figure 1.1 shows an example of an initial vertex set in the P_3 convexity in a tree, where the gray vertices are those in such a set S. We can see that the vertices v_8, v_9, v_{10}, and v_{11} are added to S at the second time step. In the same way, v_6 and v_7 are added to S at the third time step, as well as v_3 is added at the forth, and v_2 at the fifth time step. It is easy to see that $V(G) \setminus \{v_4, v_5\}$ is a convex set. We can determine all convex sets by starting from every subset of vertices and applying the same process.

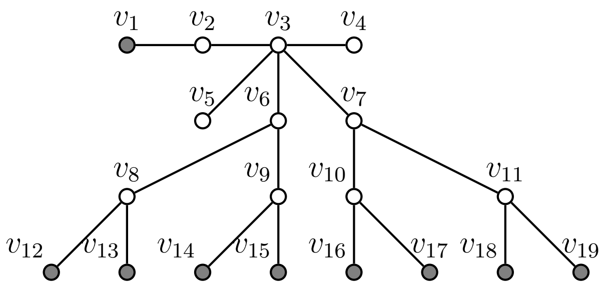


Figure 1.1: Example of an initial vertex set in a P_3 convexity.

Now, we present some convexity invariants.

We say that a graph convexity $(G, \mathcal{C})$ is an *interval convexity* when it admits a function $I : 2^{V(G)} \to 2^{V(G)}$, called *interval function*, where for every $S \subseteq V(G)$, $I(S) = S \cup W$, where W is a set of vertices determined by some property Π of the considered convexity. Let $I_0[S] = S$ and $I_t[S] = I_{t-1}[S] \cup W_t$ for any positive integer t, where W_t is the set of all vertices that can be added according to Π. For example, in Figure 1.1 we can take S as the set of the gray vertices, and $W_1 = \{v_8, v_9, v_{10}, v_{11}\}$. The notion of interval function is clear with respect to f-reversible processes as well as for the both irreversible processes considered in this thesis.

The *interval number* of G is the minimum cardinality $in(G)$ of a subset $S \subseteq V(G)$ satisfying $I_1[S] = V(G)$. We call such a set S as an *interval set* of G. In the example given by Figure 1.1 we can see that all leaves of the tree must be in all interval set. Moreover, it is enough to add three vertices in order to obtain an interval set, for example v_2, v_6, and v_7, and there is no interval set with less vertices.

The *convex hull* of a subset $S \subseteq V(G)$ is the smallest convex set $H(S)$ that contains S. The convex hull of a set S can also be denoted by $I_\infty(S)$ for interval convexities. If $H(S) = V(G)$, then S is a *hull set* of G. The cardinality $hn(G)$ of a minimum hull set of G is the *hull number* of G. In Figure 1.1, the convex hull of S given by the gray vertices is $V(G) \setminus \{v_4, v_5\}$, and since $H(S)$ is not $V(G)$, then S is not a hull set of G. We can see that any hull set of G must contain all leaves of the tree, and hence $S \cup \{v_4, v_5\}$ is a minimal hull set of G.

We say that $V(G)$ and the empty set are *trivial convex sets*. The *convexity number* of G is the largest cardinality $cx(G)$ of a non-trivial convex set $C \subset V(G)$. In Figure 1.1 we can just remove any vertex of degree one in order to obtain a non-trivial convex set of maximum cardinality.

The *Carathéodory number* is the smallest integer $c(G)$ such that for every $S \subseteq V(G)$, and every $u \in H(S)$, there is a subset $X \subseteq S$ in which $|X| \leq c(G)$, and $u \in H(X)$. Equivalently, the Carathéodory number can be defined as the smallest integer $c(G)$ such that for all $S \subseteq V(G)$,

$$H(S) = \bigcup \{H(X) \;:\; X \subseteq S,\ |X| \leq c(G)\}.$$

The remainder of this thesis is organized as follows. In Chapter 2 we present the results obtained on the study of f-reversible processes. Chapter 3 contains the results concerning the And/Or-convexity and the generalized threshold processes. Chapter 4 is devoted to problems of editing graphs by the removal of a matching. Finally, Chapter 5 contains our conclusions and open problems.

Chapter 2

f-Reversible Processes

"Ladies and gentlemen, let the 74th Hunger Games begin – and may the odds be ever in your favor!" (Claudius Templesmith)

— S. Collins, *The Hunger Games*

2.1 Dissemination on Graphs

Let V be a set defined by you, your family, your friends and coworkers that you contact directly, besides all of the relatives, friends and coworkers from the people in V, and so on. Suppose that this set will not change for a while supposed to be infinite. Let G be the *graph* whose *set of vertices* is V and *set of edges* is defined such that there exists an edge between two persons if and only if they are known. It is not difficult to see that this graph is *finite, no directed, simple, and connected* .

We can partition V into two sets: the people who eat healthily and who do not. Suppose you have the OCD (Obsessive-compulsive disorder) of counting, every day, the number of people who you know that eat healthily. Moreover, if today exist at least p_i people eating differently than you, then tomorrow you will change your eating habits, otherwise you will remain them. We assume a stronger supposition, where every person $j \in V$ has the same disorder, such that j changes its eating from one day to the next if and only if it knows at least p_j people with the opposite habits.

The above situation exemplifies an *automata network*, that is a discrete dynamical system defined over a finite set of vertices $V = \{v_1, v_2, \ldots, v_n\}$ and a finite set of *states* Q, where each vertex v_i has a state $c_i(t) \in Q$ at each integer instant $t \geq 0$. Moreover, each vertex v_i has a *state transition rule* from the instant t to $t+1$, which depends on the states of a subset of vertices in the instant t, such that $c_i(t+1) \in Q$.

More precisely, we are interested in a particular case of automata networks, that are the *dynamic systems on graphs*, where the set of vertices is the same of the graph, the state updating of a vertex is done based on its neighborhood, and are used only two states, 0 and 1.

The example above illustrates a discrete dynamical system on graphs that uses only two states. Naturally, this example does not make much practical sense for several reasons: it was not defined, and probably it does not exist, a border between what is eating healthily and what is not; the number of people is considered constant during all process; people know others, therefore new edges could be added to the graph, as well as edges can be removed over the days; each person knows the eating habit of each acquainted, at each day; all people have the same OCD is also fishy.

Let us consider a more realistic example. Suppose we are in the election season and there are two presidential candidates. Consider the same graph G as in the previous example, but containing only the people who can vote. When the electoral campaign begins, each person has a pre-candidate. Moreover let us assume that there is no white or null votes. In this case, the states are defined by the candidates. Suppose that each person i changes its vote if and only if he knows at least p_i people with opposite opinion to himself. We can use a weekly time stamp in order to be more realistic.

We can note that the last example, although it is much more plausible than the previous one, it also does not represent reality perfectly. For example, the value of p_i of an individual i represents the importance degree of others opinions over yours. However, the rule to determine your decision in each week is always the same. If this assumption were true, we would not need to have an election campaign, since the result can be obtained by simply simulating it from the first opinions of each person. In order to avoid questions of interpretation, which are extraneous to prediction, we remove any subjectivity and consider only *deterministic systems*, that is, the state updating function is always the same throughout the process.

The leader election is an important problem in the theory of distributed algorithms, multi-agent systems and in sociobiology. Through the use of *cellular automata* of two states, which are a particular case of automata networks, Peter Banda [10] has obtained significant results in this problem, showing that the approach of such systems is quite fruitful. In fact, the study of dynamic systems in graphs has wide applicability in several distinct areas, as in astrophysics and physical simulations [121, 136], simulation of biological cell population [95], modeling of chemical systems [93], social influence [36, 73, 92, 122–124], gene expression networks [91], immune systems [3], cellular automata [4], percolation [9], marketing strategies [56, 64, 92], finite discrete dynamic systems [16, 117, 137], neural networks [81], local interaction games [106, 109] and in distributed computing [71, 72, 87, 98, 111, 120]. These works show that, even eliminating some subjective questions, we still achieve significant results of prediction and analysis of the behavior of such discrete systems.

Other opinion dissemination models have also been proposed for the study of more specific situations, where the dissemination among the individuals can be modified in such a way that *clusters* among similar individuals are formed according to some property and as the process follows. Examples can be given by hemophilia dissemination [6, 7], or using *stochastic actor models* [101, 105, 132, 133].

2.2 Reversible Processes

In this thesis we concern with a particular case of dynamic systems on graphs, that are the *reversible processes*. A reversible process is defined over a simple, finite and non-directed graph $G = (V, E)$, where each vertex has one state in $\{0, 1\}$, and a function $f : V(G) \to \mathbb{N}$, called *threshold function*. At each discrete time step $t \geq 0$, each vertex $v \in V$ changes its state if and only if v has at least $f(v)$ neighbors with opposite state to itself at t. Moreover, the state modification is done in a *synchronous way*, that is, all vertices apply the threshold function to its own state and the states of its all neighbors at each time step.

2.2.1 f-Reversible Processes

As we saw, reversible processes models iterative voting mechanisms or consensus in a network. Mustafa and Pekeč [111] considered reversible processes on graphs G such that the threshold function is constant and equals $\left\lfloor \frac{|V(G)|}{2} \right\rfloor + 1$. Such processes are known in the literature as *majority processes* or *iterative voting processes*. They were addressed to different problems in distributed computing, such as fault and em tolerance and fault recovery [87, 98, 112, 119, 120]. The reversible processes with constant threshold function and equals $k \geq 2$ are called *k-reversible processes*.

A *configuration* in a reversible processes over a graph G is a $\{0, 1\}$ state assignment to the vertices of G at t. Given a reversible process on G, a configuration at the time $t \in \mathbb{N}$ is denoted by c_t. We can note that reversible processes reach periodic behavior, since we consider the time as infinite and the number of possible configurations is finite as well as G. In this way, it is interesting from the computational point of view to know the length of the periodic phase and the number of time steps required to reach such a periodic phase. For example, in asynchronous distributed systems, where the components of the computation do not have information about the whole system and does not exist a common time stamp to all of them, it is essential to know if the system is in a *periodic configuration*, that is, if new computations cannot be formed. This property is important in the cases of the system acquire a global termination or some of its part reaches a *deadlock* behavior or even if it is the moment to start a new phase of the computation.

Given a *initial configuration* c_0, the set of configurations starting in c_0 until the last one preceding the periodic behavior defines the *transient* of the reversible process. We denote the length of the transient by $\tau(c_0)$. Note that the length of the transient can be zero, if the initial configuration is periodic. Moreover, the length of the transient and of the period of a reversible process on G could be exponential in the number of vertices of G, since there exists $2^{|V(G)|}$ possible configurations. Independently, Poljak and Sura [122], and Goles and Olivos [82], however, showed that the period has length at most 2. Moreover, in the same work, Goles and Olivos showed that the maximum transient length is polynomially upper bounded.

In Oliveira's [115] dissertation, it is done a combinatorial study on k-reversible processes based on the work of Goles and Olivos [82]. In that dissertation is presented an *energy function* that captures with more intuition the dynamics of k-reversible processes. Goles and Olivos also used an energy function in their results, but that are used in more general dynamic processes than the reversible one. Such processes are known as *threshold processes* [82, 84]. The results presented in these studies are based on more general bounds that use an algebraic approach. The new and more specific energy function, Oliveira proved in an alternative and simpler way that k-reversible processes have period upper bounded by 2, besides a better upper bound for the transient length.

Another important question related to the study of reversible processes is to determine the size of a minimum *f-conversion set* (or k-conversion set when f is a k constant function), which is a vertex set that has initial state equals 1 and whose periodic configuration is unique, where all vertices have state equal to 1. Dreyer [65] showed many results regarding this question, but also restricted to k-reversible processes. He showed that determining the size of the minimum conversion set of k-reversible processes is NP-hard for $k \geq 3$, and determined exact values for some graph classes.

2.2.2 Formalization

In this section we present the formalization of the reversible processes considered in this work and some of their characteristics, such as the periodicity after a transient phase, given by an initial configuration, besides some problems concerning such processes.

Let $G = (V, E)$ be a simple finite non-directed graph with n vertices and m edges. To each vertex of G we associate one *state* between two, that are represented by the values 0 and 1. A *configuration* at the discrete time $t \geq 0$, denoted by c_t, is an assignment of the states 0 or 1 to the vertices of G, $c_t : V(G) \to \{0, 1\}$.

A *discrete dynamical process on* G is an infinite sequence $\mathcal{P} = (c_t)_{t\in\mathbb{N}} = (c_0, c_1, \dots)$ of configurations. We say that c_0 is the *initial configuration* of $\mathcal{P}$ and $c_t(v)$ denotes the state of v at time $t \in \mathbb{N}$.

In this work, we consider a special kind of iterative processes on simple finite non-directed graphs that generalizes the majority voting approach studied by Peleg [120]. Formally, the update rule is such that c_{t+1} is obtained from c_t according to a *threshold function* $f : V(G) \to \mathbb{N}$ applied to each vertex $v \in V(G)$ and at each time $t \in \mathbb{N}$.

The update rule is defined as

$$c_{t+1}(v) = \begin{cases} 1 - c_t(v) & \text{, if } |\{u \in N(v) : c_t(u) \neq c_t(v)\}| \geqslant f(v); \\ c_t(v) & \text{, otherwise.} \end{cases} \tag{2.1}$$

In other words, every vertex v changes its state if and only if it has at least $f(v)$ neighbors with the opposite state, at each time step $t \in \mathbb{N}$. Moreover, the state updates are done synchronously.

We say that $f(v)$ is the *threshold value* of v, which does not change during the entire process. Given an initial configuration c_0, we call such an iterative process following Equation (2.1) as an *f-reversible process on* G, denoted by $R_f(G, c_0) = (c_t)_{t\in\mathbb{N}}$, or simply $R_f(G, c_0)$. If f is a k-constant function, then it is denoted by $R_k(G, c_0)$ and it is called *k-reversible process on* G, $k \in \mathbb{N}$.

Gargano *et al.* [78] considered the influence diffusion in social networks in which some individuals are "actives" to influence their neighbors for a limited number of time steps, once they were influenced by a sufficient number of neighbors. However, if a vertex has been influenced then it remains in this state forever. An f-reversible process represents the extreme case such that the elements have no "memory". Thus, the vertices do not keep themselves influenced if they have the required amount of neighbors to change their states. Moreover, each vertex needs to be convinced by a subset of neighbors to change its opinion, at each time step.

It is clear that an f-reversible process is uniquely determined by G, f and c_0. Furthermore, due to Equation (2.1), every vertex v depends on only its own state and on the states of its neighbors to define its next one. Thus, we will consider only connected graphs in the remainder of the chapter. Moreover, if $f(v) > d(v)$ for some vertex v then its initial state does not change during the whole process. Therefore we can assume $f(v) = d(v) + 1$ in this case. Hence, only threshold functions f satisfying $\text{Im}_f = \{0, \dots, \Delta(G) + 1\}$ will be considered. Note that if $f(v) = 0$ then the state of v changes at every time step $t \in \mathbb{N}$.

2.2.3 Threshold Networks

We can define a *threshold network* through a square matrix A with dimensions $d \times d$, and a *threshold vector* b of dimension d. We represent a threshold network with matrix A and threshold vector b by (A, b). Such processes model dynamic systems on graphs, where each vertex has one between two states, non necessarily equal to 0 or 1, constituting configurations c_t at each discrete time step $t \geq 0$. The state updating rule in these processes from instant t to $t+1$ is as follows:

$$c_{t+1}(v_i) = \begin{cases} 1 & \text{, if } \sum_{j=1}^{d} A_{ij} c_t(v_j) - b_i \geqslant 0; \\ 0 & \text{, otherwise.} \end{cases} \tag{2.2}$$

In fact, the state updating of the vertices is done by comparing the vector Ac_t and b, for each index $1 \leq i \leq d$, considering c_t as a vector, where its i-th element is given by $c_t(v_i)$.

Next we show that$f$-reversible processes can be modeled as threshold networks, that is, f-reversible processes are particular cases of threshold networks. In fact, we do it for (f_1,f_2)*-reversible processes*, such that there are two threshold functions f_1 and f_2, where v changes its state from 0 to 1 according to f_1, and from 1 to 0 according to f_2. If $f = f_1 = f_2$, then we obtain an f-reversible process. The proof is analogous to that presented by Dreyer [65] in his thesis for k-reversible processes and it can be found in Appendix 5.1.5.

Theorem 2.1 *An (f_1,f_2)-reversible process on a graph G is a threshold network (A, f_1), such that*

$$A_{ij} = \begin{cases} 1 & \text{, if } v_i v_j \in E(G) \text{ and } i \neq j; \\ 0 & \text{, if } v_i v_j \notin E(G) \text{ and } i \neq j; \\ f_1(v_i) + f_2(v_i) - d(v_i) - 1 & \text{, if } i = j. \end{cases}$$

Proof. Let $n_t^1(v_i)$ and $n_t^0(v_i)$ be the number of neighbors of v_i with states 1 and 0 at time t, respectively. By Equation 2.2 It is enough to show that $c_{t+1}(v_i) \neq c_t(v_i)$ if and only if v_i has at least $f_1(v_i)$ neighbors with state 1, or v_i has at least $f_2(v_i)$ neighbors with state 0 at time t, when v has states 0 and 1, respectively.

- $c_t(v_i) = 0$: we get that $c_{t+1}(v_i) = 1$ if and only if $\sum_{j=1}^{n} A_{ij} c_t(v_j) - b_i \geqslant 0$. Hence:

$$\begin{aligned} \sum_{j=1}^{n} A_{ij} c_t(v_j) &\geqslant f_1(v_i); \\ (f_1(v_i) + f_2(v_i) - d(v_i) - 1) c_t(v_i) + n_t^1(v_i) + n_t^0(v_i) &\geqslant f_1(v_i); \\ n_t^1(v_i) &\geqslant f_1(v_i). \end{aligned}$$

In this way, v_i changes its state from 0 to 1 if and only if $f_1(v_i) \leq n_t^1(v_i)$, as (f_1,f_2)-reversible processes.

- $c_t(v_i) = 1$: we get that $c_{t+1}(v_i) = 0$ if and only if $\sum_{j=1}^{n} A_{ij}c_t(v_j) - b_i < 0$. Hence:

$$\begin{aligned}
\sum_{j=1}^{n} A_{ij}c_t(v_j) &< f_1(v_i);\\
(f_1(v_i) + f_2(v_i) - d(v_i) - 1)c_t(v_i) + n_t^1(v_i) + n_t^0(v_i) &< f_1(v_i);\\
f_1(v_i) + f_2(v_i) - d(v_i) - 1 + n_t^1(v_i) &< f_1(v_i);\\
f_2(v_i) - (n_t^1(v_i) + n_t^0(v_i)) - 1 + n_t^1(v_i) &< 0;\\
f_2(v_i) - n_t^0(v_i) - 1 &< 0;\\
f_2(v_i) &< n_t^0(v_i) + 1;\\
f_2(v_i) &\leqslant n_t^0(v_i).
\end{aligned}$$

Analogously, v_i changes its state from 1 to 0 if and only if $f_2(v_i) \leq n_t^0(v_i)$, as (f_1,f_2)-reversible processes. □

Such processes simulate situations in which there exists a preference for acceptance of one opinion over another. Oliveira, Barbosa and Protti [115] proved that k_1-k_2-reversible processes (when f_1 and f_2 are k_1 and k_2 constant functions, respectively) are particular threshold networks.

An *irreversible process* is one that a vertex does not change its state once reached state 1. Among their many applications, we can cite the opinion dissemination in a network, such that the interest is to acquire the largest possible number of people, and using a few number of opinion transmitting agents, vertices of state 1, or by the necessity of a quick spread. Disease spread is another important application, where it is desired to measure its reach in a given population. In short, these processes are applied to dynamic systems that there is no meaning the state change, once an individual is reached.

Irreversible processes are also particular cases of threshold networks. In order to demonstrate this fact, we only consider the first case in the proof of Theorem 2.1 and same threshold network (A, f_1).

With Theorem 2.1, we can use the knowledge on threshold networks [82] for f-reversible processes.

2.2.4 The Periodic Behavior

Let $R_f(G, c_0)$ be an f-reversible process. We say that a configuration c_t *reaches* all of the configurations $c_{t'}$, for every $t \geq 0$ and $t' > t$, or that c_t is *reached* by all configurations $c_{t''}$, for every $t \geq 1$ and $0 \leq t'' < t$.

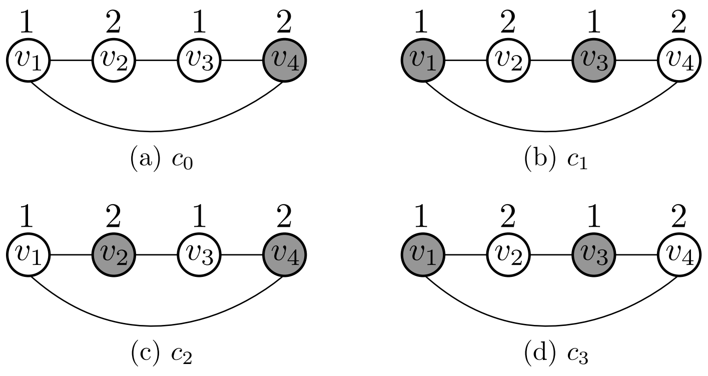


Figure 2.1: Example of the dynamic of an f-reversible process.

Given the configuration c_t of $R_f(G, c_0)$, $t \geq 1$, we say that c_{t-1} is a *predecessor configuration* of c_t and that c_t is the *successor configuration* of c_{t-1}. If the successor of a configuration c is itself, then c is known as *fixed point configuration.*

Given an f-reversible process $R_f(G, c_0)$, we call as *phase graph* of $R_f(G, c_0)$ the directed graph whose vertex set is given by all configurations $(c_t)_{t \in \mathbb{N}}$, that is, all reachable configurations from c_0. Moreover, there exists a directed edge from c toward c' if and only if c is predecessor of c'. We denote the phase graph of $R_f(G, c_0)$ by $F(G, c_0, f)$.

Figure 2.1 exemplifies the execution of an-f-reversible process over a C_4 whose vertex set is $\{v_1, v_2, v_3, v_4\}$. The gray vertices have state equal to 1. The number over each vertex v_i represents the value $f(v_i)$. The figures of this chapter will follow this pattern. Figure 2.2 represents the phase graph of the process presented in Figure 2.1.

By Figure 2.1, we can see that, in c_0, v_1 and v_3 have a neighbor with opposite state to each one and threshold value 1. Moreover, v_4 has two neighbors with opposite state to itself and threshold value equals 2, while v_2 has no neighbor with opposite state to itself and has threshold value equals 2. Therefore, only v_2 does not change its state from the time step 0 to 1. The same analysis can be done from the time step 1 to 2, and so on.

Let $R_f(G, c_0)$ be an f-reversible process. Since G is finite and c_{t+1} is obtained deterministically from c_t, according to Equation (2.1), the number of possible configurations equals 2^n. Hence, there must exist a finite time step when the process becomes periodic. The set of configurations preceding the periodic phase is called

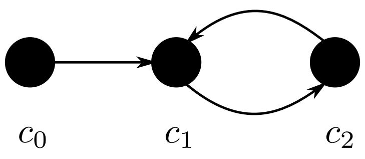


Figure 2.2: Phase graph of the process presented in Figure 2.1.

transient, and its length is denoted by $\tau(c_0) \geq 0$. The length of the periodic phase is called *period*, and it is denoted by $p(c_0) \geq 1$. A *periodic configuration* is one that occurs in the periodic phase. Note that the first one is reached at time $\tau(c_0)$. Formally, the period and transient length satisfy the following conditions:

- $c_{t+p(c_0)} = c_t$, for all $t \geq \tau(c_0)$;
- $c_{t+q} \neq c_t$, for all $(t < \tau(c_0)$ and $q \geq 1)$ or $(t \geq \tau(c_0)$ and $1 \leq q < p(c_0))$.

Two natural parameters arise from the above definitions. Given a threshold function f and a graph G, denote the largest transient length over all initial configurations by $\tau_f(G)$ (resp. $\tau_k(G)$ if f is a k-constant function). Analogously, let $p_f(G)$ be (resp. $p_k(G)$ if f is a k-constant function) the largest period over all initial configurations.

Figure 2.3 shows the phase graphs over all of the 2^4 possible configurations with respect to the f-reversible process described up to now. The subscribed in each vertex denotes the configuration $c_{x_1x_2x_3x_4}$, $x_i \in \{0,1\}$, where x_i represents the state of the vertex v_i, $1 \leq i \leq 4$.

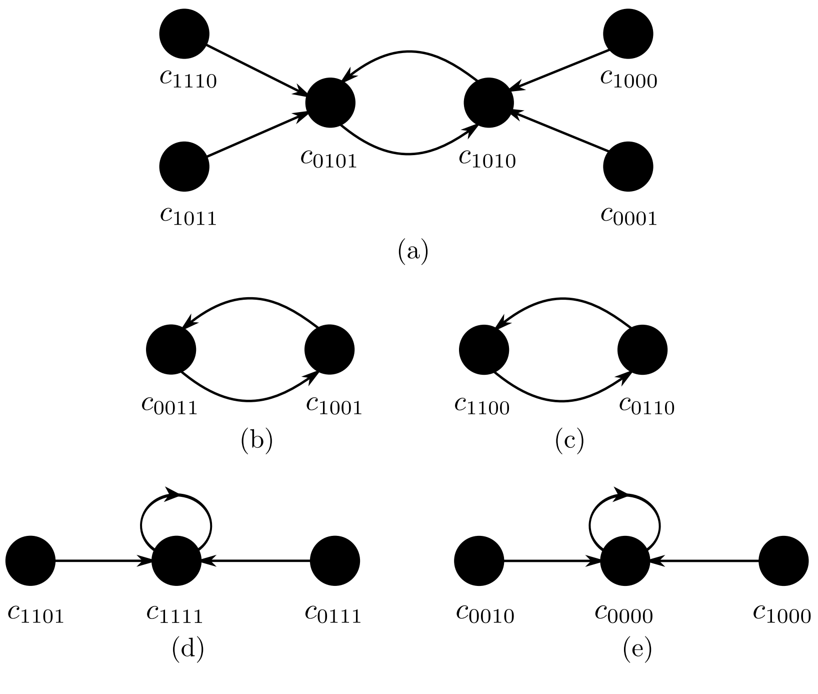


Figure 2.3: Phase graphs of the f-reversible process defined in Figure 2.1.

We can note by Figure 2.3 that starting the process in any configuration among the 2^4 possible ones it always reaches a directed cycle. In Figures 2.3d and 2.3e the period is equal to 1 in each one, while the remainder phase graphs finish in cycles of length 2. Moreover, the transient length in Figures 2.3b and 2.3c are equal to 0, while all the other initial configurations have transient length equal to 1.

Consider now Figure 2.4. We can see that the first periodic configuration occurs only at the time step 3, that is, the transient length is equal to 3, while its period values 2. In fact, Figure 2.4 shows that there exists an f-reversible process with unbounded transient length, since such processes over a P_n have transient length equals $n-2$. Figure 2.5 represents its phase graph.

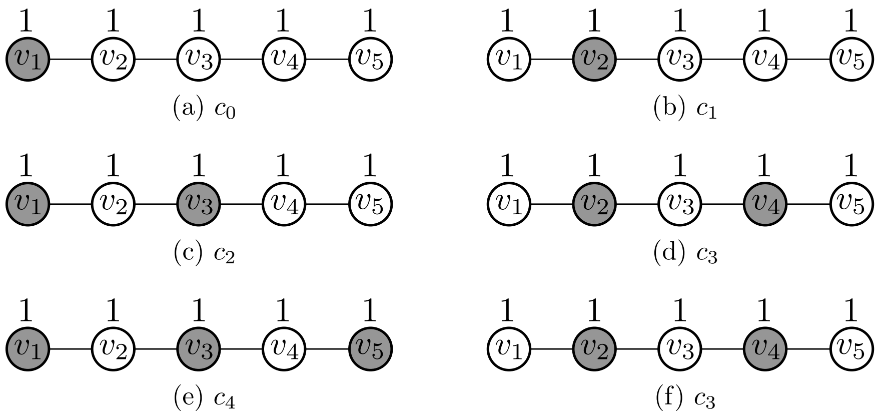


Figure 2.4: Example of an f-reversible process with unbounded transient length.

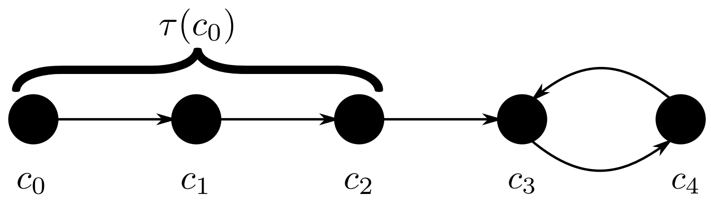


Figure 2.5: Phase graph of the process described in Figure 2.4.

We can observe that the period in Figure 2.4 is equals 2, even with an unbounded transient length. In fact, Goles, Fogelman and Pellegrin [84] showed the following result:

Theorem 2.2 *[84] Let (A, b) be a threshold network. If A is symmetric, then the period length of (A, b) is at most 2, for any initial configuration.*

Combining Theorem 2.1 and Theorem 2.2, where is used a symmetric matrix in the proof, we can obtain the following corollary:

Corollary 2.3 *The period of (f_1,f_2)-reversible processes is at most 2.*

The same result has been obtained by Oliveira, Barbosa and Protti [115] for k-reversible processes, k-irreversible processes and k_1-k_2-reversible processes. Analogously the same result works for irreversible processes. However, note that Corollary 2.3 cannot be applied to threshold networks (A, b) whose matrix A is not symmetric. For example, considering directed graphs where a vertex changes its state if and only if it has at least k in-neighbors with opposite state, then Corollary 2.3 does not work. Figure 2.6 exemplifies such a negative result, where each vertex has

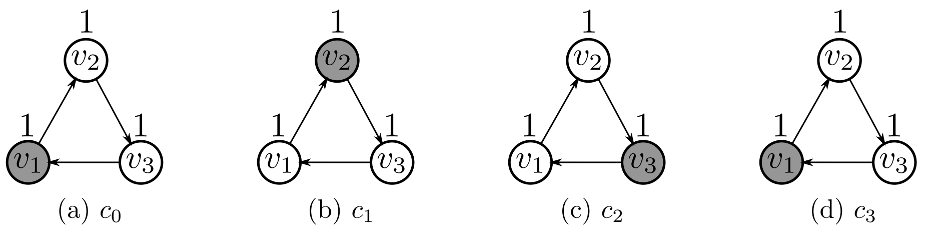


Figure 2.6: Example of threshold network with period greater than 2.

exactly an in-neighbor and $k = 1$. It is not difficult to verify that $p(c_0) = n$ for such a process.

Goles, Fogelman and Pellegrin [84] also determined a polynomial upper bound for the maximum transient length of a threshold network (A, b), denoted by $\tau(A, b)$:

Theorem 2.4 *[84] Let (A, b) be a threshold network on a graph G. If A is symmetric, then $\tau(A, b) \leq \sum_{i=1}^{n} \sum_{j=1}^{n} |A_{ij}| + 2 \sum_{i=1}^{n} |b_i|$.*

Corollary 2.5 *The maximum transient length of an f-reversible process on a graph G is upper bounded by $2m + 2n\Delta(G)$.*

Proof. By Theorem 2.4 and using a matrix A as in Theorem 2.1 and threshold vector equals 0, it follows that:

$$T(A, b) \leq 2m + \sum_{i=1}^{n} |2f(v_i) - d(v_i) - 1|$$

The maximum value of $|2f(v_i) - d(v_i) - 1|$ is acquired when $f(v_i) = \Delta(G) + 1$ and $d(v_i) = 1$, for every $1 \leq i \leq n$. Therefore, the expression inside the modulo is positive, and it follows that:

$$\begin{aligned} T(A, b) &\leq 2m + \sum_{i=1}^{n} |2f(v_i) - d(v_i) - 1| \\ &= 2m + \sum_{i=1}^{n} (2(\Delta(G) + 1) - 1 - 1) \\ &\leq 2m + \sum_{i=1}^{n} (2\Delta(G)) \\ &= 2m + 2n\Delta(G) \end{aligned}$$

□

Corollary 2.5 shows that the maximum transient length of f-reversible processes is polynomially upper bounded with relation to the input graph, although there is an exponential amount of configurations. However that is not a tight upper bound. This can be verified observing that at least one vertex v_i must be such that $f(v_i) \leq \Delta(G)$, otherwise the initial configuration is already periodic, and hence $\tau(A, b) = 0$. Moreover, since we consider only simple connected graphs, if $d(v_i) = 1$ for all vertices, then $G = K_2$. In this case it is possible to verify that $\tau(A, b) \leq 1$ for any initial configuration, where the maximum value is obtained for example in the case that $f(v_1) = 0$, $f(v_2) = 1$, and $c_0(v_1) = c_0(v_2)$. However, by the above upper bound, we get that $\tau(A, b) \leq 2m + 2n\Delta(G) = 4$.

With this result it is possible to know in polynomial time if an f-reversible process has period equals 1 or 2 by executing the transitions of the configurations throughout the process. Such a complexity can be given since we know that there are at most $O(m + n\Delta(G))$ time steps to reach a periodic configuration, by Corollary 2.5, and each configuration updating can be done in $O(n+m)$. Hence there exist a general algorithm whose complexity is $O(m(n + m) + n\Delta(G)(n + m)) = O(m^2 + n^2m)$.

Despite the existence of a polynomial time algorithm to determine the size of the period of an f-reversible process, an interesting question is to find conditions that, only from the initial configuration, it is possible to determine the size of the period, without the need to execute the process step by step.

2.2.5 The Potential Function

The proof of Theorem 2.2 of Goles and Olivos [81] was better presented by Goles, Fogelman, and Pellegrin [84], where they used an algebraic monotonic operator called *energy function*. Given a threshold network (A, b) and a configuration c_t, the energy function is given as follows:

$$E(c_t) = -\frac{1}{2}\sum_{i=1}^{n} c_t(v_i) \sum_{j=1}^{n} A_{ij} c_t(v_j) + \sum_{i=1}^{n} b_i c_t(v_i) \tag{2.3}$$

This definition is very similar to that of the energy function associated with *Hopfield networks* [90]. It is a Lyapunov function and can be used to prove several results associated with the period and transient lengths of threshold and majority networks. This function dates back to Ref. [84] (page 269, inside the proof of Proposition 2) and was later reproduced in Ref. [83] (page 70, Eq. (3.3)).

The energy function 2.3, although useful for showing upper bounds to the transient and period lengths of general threshold networks, it does not capture the necessary intuition to achieve fair limits on specific processes, such as f-reversible processes.

We present the new energy function defined by Oliveira, Barbosa, and Protti [115] for k-reversible processes, which is more intuitive than Equation 2.3 when applied to such processes. Before present it, let give some definitions. For all $t \in \mathbb{N}$, let $S_1(t)$ and $S_2(t)$ be a bipartition of $V(G)$, where $S_1(t)$ denotes the set of vertices that change their states at time t and $S_2(t)$ the set of those that do not. Given a vertex v, we denote the number of neighbors with the opposite state of v

at t by $op_t(v)$. Thus, it follows that

$$S_1(t) = \{v : op_t(v) \geq k\}$$

and

$$S_2(t) = \{v : op_t(v) < k\}.$$

Such an energy function is based on the difference $op_t(v) - f(v)$, for every vertex $v \in S_1(t)$, and in the difference $f(v) - op_t(v)$, for every vertex $v \in S_2(t)$. These differences represent the facility of a vertex of $S_1(t)$ to change its state, besides what is necessary, and the facility of a a vertex of $S_2(t)$, also besides what is necessary, to keep its state, respectively. The energy function is given as follows:

$$E(t) = \sum_{v \in S_1(t)} (op_t(v) - k) + \sum_{v \in S_2(t)} (k - op_t(v)) \tag{2.4}$$

We slightly modify Equation 2.4, such that it can be used to general f-reversible processes. We call it as *potential function*. For this, we also slightly modify the definitions of $S_1(t)$ and $S_2(t)$ such that

$$S_1(t) = \{v : op_t(v) \geq f(v)\}$$

and

$$S_2(t) = \{v : op_t(v) < f(v)\}.$$

The potential function is as follows:

$$P(t) = \sum_{v \in S_1(t)} (op_t(v) - f(v)) + \sum_{v \in S_2(t)} (f(v) - op_t(v)) \tag{2.5}$$

Clearly the potential function does not assume negative values, since it is formed by the sum of non-negative portions given in the definition of sets $S_1(t)$ and $S_2(t)$. In addition, the set $S_2(t)$ always contributes positively, unless it is empty, contributing nil. Figure 2.7 shows an example of an f-reversible process, while Table 2.1 shows the number of neighbors with opposite state of each vertex and in each configuration, besides the value of the energy function given by Equation 2.4:

Fig. 2.9 contains plots of the potential and energy functions against time. One of them (filled line) is the energy function represented by Equation 2.3. The other (dotted line) refers to the potential function of Equation 2.5. Data in the plots correspond to the tree depicted in Figure 2.8. Both plots refer to a 2-reversible

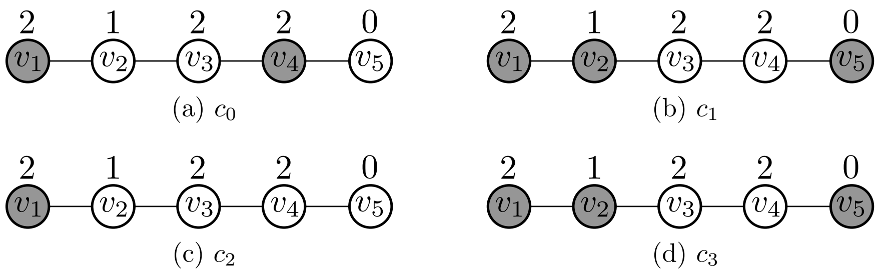


Figure 2.7: Configurations of an f-reversible process over P_5.

Table 2.1: Sets $S_1(t)$, $S_2(t)$, values $op_t(v_i)$, $1 \leq i \leq 5$, and potential function $P(t)$.

Config.	$S_1(t)$	$S_2(t)$	$op_t(v_1)$	$op_t(v_2)$	$op_t(v_3)$	$op_t(v_4)$	$op_t(v_5)$	$P(t)$
c_0	$\{v_2, v_4, v_5\}$	$\{v_1, v_3\}$	1	1	1	2	1	3
c_1	$\{v_2, v_5\}$	$\{v_1, v_3, v_4\}$	0	1	1	1	1	5
c_2	$\{v_2, v_5\}$	$\{v_1, v_3, v_4\}$	1	1	0	0	0	5
c_3	$\{v_2, v_5\}$	$\{v_1, v_3, v_4\}$	0	1	1	1	1	5

process and, in the case of the energy function, to $b_i = 1.5$ as the additional required parameter for every vertex v_i. The initial configuration has 0 at vertices v_2–v_{11} and 1 at all the others. The two functions differ markedly and no simple reduction seems to exist to transform one into the other. In particular, the potential function is nondecreasing (rather than monotonically decreasing), as illustrated by Fig. 2.9.

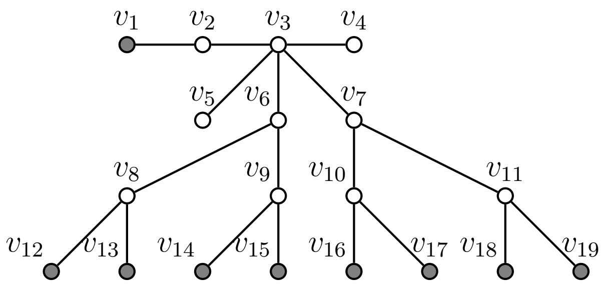


Figure 2.8: The initial configuration of a 2-reversible process on a tree whose gray vertices have state 1.

2.2.6 The Minimum f-Conversion Set Problem

In this subsection we deal with another problem related to f-reversible processes, where are considered only those having a fixed point as their periodic configuration.

We say that an f-reversible process $R_f(G, c_0)$ is *uplifting* if all vertices achieve state 1 in a finite number of time steps, which is given by $\tau(c_0)$. In addition, the set of vertices with state equal to 1 in the initial configuration is called the *f-conversion set* of G. Clearly, for an f-conversion set to exist, all of the vertices must have

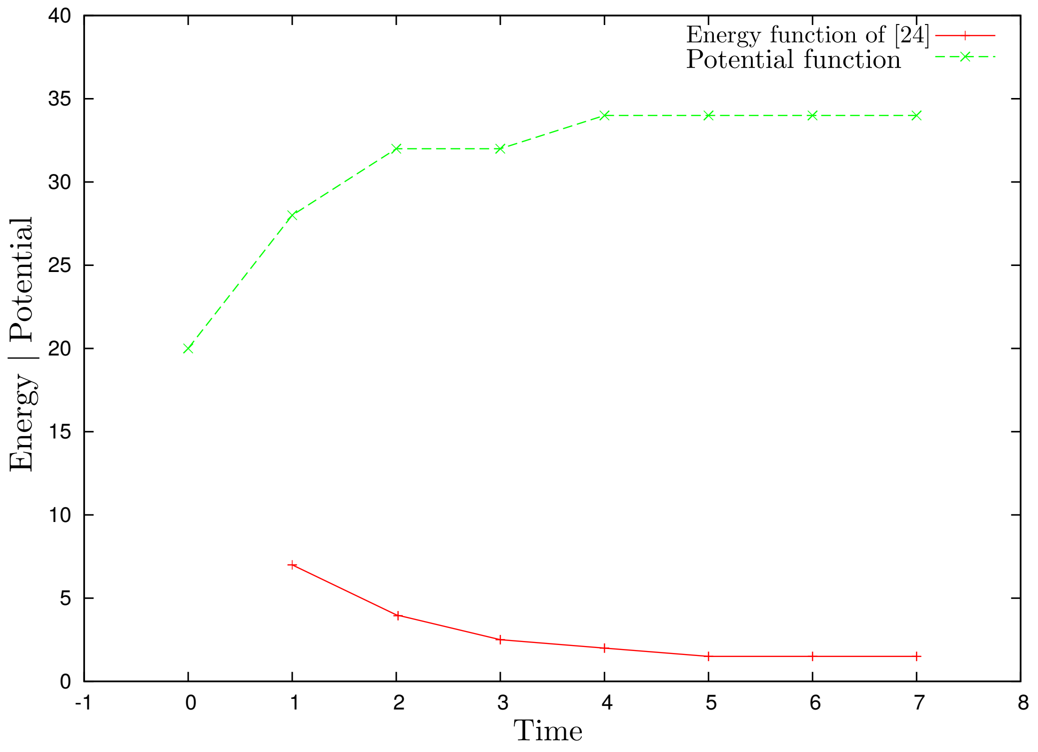


Figure 2.9: Time evolution of the potential (dotted line) and energy (filled line) functions.

a positive threshold value. In addition, for any vertex v whose threshold value is $d(v)+1$, we get that the initial state of v must be equals 1. Thus, we will restrict ourselves to specific f-reversible processes, where $f : V(G) \to \{1, 2, \ldots, \Delta(G)+1\}$, and $c_0(v) = 1$ for all v such that $f(v) > d(v)$, and whose initial configurations lead to be uplifting.

By the above restrictions, we can see that there are two trivial f-conversion sets for a graph G, which are the set $V(G)$, and that process where $f(v) \geq d(v)+1$ for every vertex v. Figure 2.10 shows the complete execution of a 2-reversible process, where we can see that it is uplifting, besides exemplifying a non-trivial f-conversion set given by $\{V_1, v_3, v_5, v_7\}$:

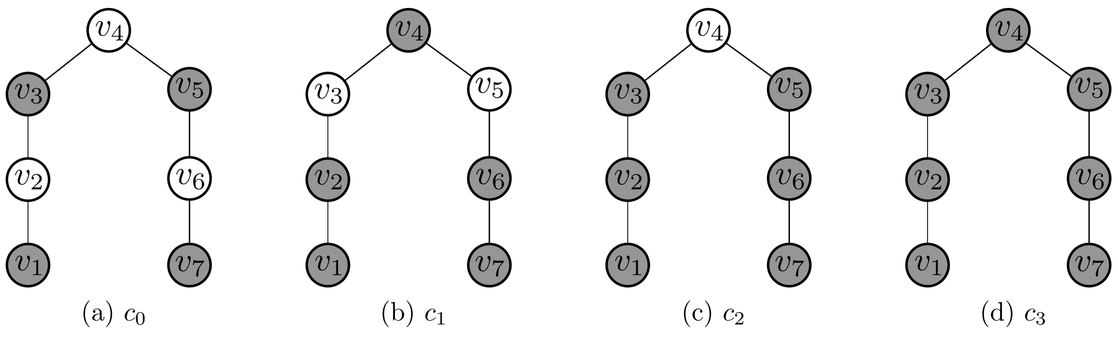
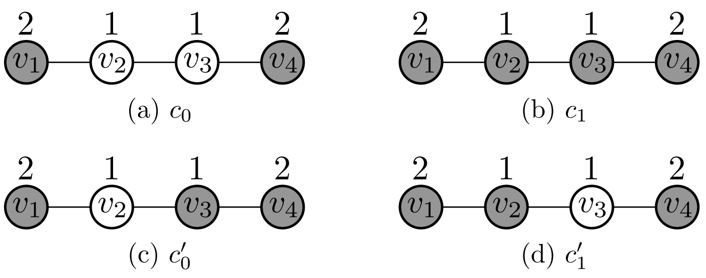


Figure 2.10: Example of a non trivial f-conversion set.

f-conversion sets are defined not only for $f$reversible processes, but also for irreversible processes. Given a graph G and threshold values for each vertex, we can see that a set C is uplifting in an f-reversible process over G is also an f-conversion set in the irreversible process over G, which uses the same threshold values. This is due to the fact that at each time step, the set of vertices with equals 1 in the f-reversible process is always contained in the set of vertices with state equals 1 in the irreversible process. Moreover, since C uplifts all the vertices to state 1 at the end of the f-reversible process, so does the irreversible process. In addition, given an f-conversion set C of G in an irreversible process, every superset of C is also an f-conversion of G. The same is not true for f-reversible processes, as we can see by Figure 2.11.



(a) c_0 (b) c_1 (c) c'_0 (d) c'_1

Figure 2.11: Example of a superset that is not an f-conversion set of G.

Figures 2.11a and 2.11c represent distinct initial configurations over the same graph and threshold function. We can observe that $\{v_1, v_4\}$ is an f-conversion set, but its superset $\{v_1, v_3, v_4\}$ is not, where vertices v_2 and v_3 alternate their states indefinitely. This fact exemplifies that f-reversible processes have a more complex analysis related to the irreversible ones.

An interesting question is to answer if a given initial configuration causes the f-reversible process to uplift just by looking at the properties of the graph and the threshold values. This question is not included in determining the size of the period of f-reversible processes without the need to execute the process at each time step. This is true because, besides determining that the period is 1, we must show that all vertices have state equal to 1 in the periodic configuration. The following remark shows a sufficient condition for an f-reversible process not to reach fixed point.

Observation 2.6 *Let an f-reversible reversible process $R_f(G, c_0)$ and a bipartition of $V(G)$ into sets A and B, such that each vertex $v \in A$ has opposite state to each vertex $u \in B$, at $t \geq 0$, and $|N_B(v)| \geq f(v)$, for every vertex $v \in A$, and $|N_A(u)| \geq f(u)$, for every vertex $u \in B$. We can observe that this situation causes a periodic behavior with period exactly 2, where A and B alternate their states for every instant $t' \geq t$, independently of the states of the other vertices of G. Note that c_t is not necessarily periodic.*

An example of configuration satisfying Observation 2.6 can be done by the K_2 where each vertex has threshold value equals 1 and they have opposite states, as vertices v_2 and v_3 in Figure 2.11c. Figure 2.12 presents an non-trivial example:

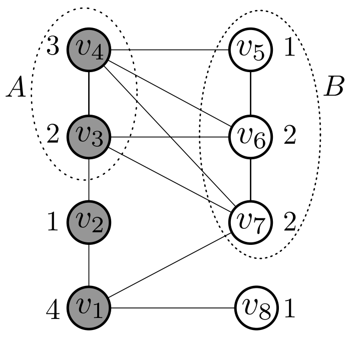


Figure 2.12: Example of configuration satisfying Observation 2.6.

We will deal with the problem of finding the size of an smallest f-conversion set, denoted by $r_f(G)$. If f is a constant function with image $0 < k \leq \Delta(G) + 1$, we denote the such a parameter as $r_k(G)$. In other words, we want to find an initial configuration that has the smallest number of vertices with state 1, so that the process uplifts. We consider the following decision problem:

f-CONVERSION SET
Input: A simple, finite, undirected, and connected graph G, a threshold function $f : V(G) \to \mathbb{N}$, and an integer $q \geq 1$.
Question: Is $r_f(G) \leq q$

Note that in the definition of f-CONVERSION SET we require the graph to be connected, since vertices in distinct connected components do not influence the dynamics of state change of each other. Therefore $r_f(G) = \sum_{i=1}^{w(G)} r_f(W_i)$, where W_i is the i-th connected component of G.

It is not hard to verify that Figure 2.10a and Figure 2.11a are examples of initial configurations where the sets of vertices with state 1 have minimum size. In fact, for 2-reversible processes on trees T with ℓ leaves, Dourado *et al.* [59] showed that $r_2(T) \leq \frac{n+\ell}{2}$. If $\ell = 2$, that is, T is a path, then the previous limit is fair, as we can see in Figure 2.11. Independently, Dreyer [65] and Dourado *et al.* [59] reached the same bound for paths and also showed that it is the same for cycles, for 2-reversible processes.

Dourado *et al.* [59] showed the NP-hardness of determining $r_2(G)$ for general graphs G. They also stated an algorithm based on dynamic programming which computes $r_f(P_n)$ for specific paths P_n with n vertices. They consider both threshold values 1 and 2, where every v with $f(v) = 1$ has a neighbor u with $f(u) = 1$. Thus, computing $r_f(G)$ even for paths and cycles remains open.

In his thesis, Dreyer [65] proved that f-Conversion Set is NP-complete for k-reversible processes on k-regular graphs, $k \geq 3$. Moreover he gave exact values of $r_k(G)$ for some specific graph classes. Dourado *et al.* [59] proved that determining $r_2(G)$ is NP-hard. Table 2.2 presets a resume of the studied cases in the literature related to f-Conversion Set. However, all of these results are restricted to k-reversible processes. Symbol $G_{p,q}$ denotes the *complete grid graph* with dimensions p and q. Symbol $C_{p,q}$ denotes the *circular graph*, that is the complete grid graph where its joined the horizontal ends of the same line. Finally, symbol $T_{p,q}$ denotes the *torus graph*, that is the circular graph where we join the vertical ends of the same column.

Table 2.2: Resume of the main exact results on $r_f(G)$.

Graph G	$r_k(G)$	Reference
P_n and C_n	$r_2(P_n) = r_2(C_n) = \left\lfloor \frac{n+2}{2} \right\rfloor$	Dreyer [65] and Dourado *et al.* [59]
$K_{p,q}$, $p \geq q$	$r_k(K_{p,q}) = \begin{cases} p+n & \text{, if } p < k \\ p & \text{, if } p \geq k \text{ and } n < k \\ n+1 & \text{, if } p < 2k-1 \text{ and } n \geq k \\ p+n-2(k-1) & \text{, if } p \geq 2k-1 \text{ and } n \geq k \end{cases}$	Dreyer [65]
$G_{p,q}$	$r_2(G_{2,q}) = q+1$ $r_3(G_{2,q}) = q+2$ $r_4(G_{p,q}) = 2p+2q-4+\lfloor (p-2)(q-2)/2 \rfloor$ $r_3(G_{p,q}) = \begin{cases} (3q+1)/2 & \text{, if } p \text{ or } q \text{ is odd} \\ (3q+2)/2 & \text{, if } n \text{ is even} \end{cases}$	Dreyer [65]
$T_{p,q}$	$r_3(T_{p,q}) = \begin{cases} (3q+1)/2 & \text{, if } p \text{ or } q \text{ is odd} \\ 3q/2 & \text{, if is even and } n \geq 6 \\ (3q+2)/2 & \text{, if } n = 2, 4 \end{cases}$ $r_4(T_{p,q}) = \begin{cases} max\{q \left\lfloor \frac{p}{2} \right\rfloor, p \left\lfloor \frac{q}{2} \right\rfloor\} & \text{, if } p \text{ or } q \text{ is odd} \\ \frac{pq}{2} & \text{, otherwise} \end{cases}$	Dreyer [65]
$C_{p,q}$	$r_2(C_{2,q}) = \begin{cases} q+1 & \text{, if } q \text{ is odd} \\ q+2 & \text{, otherwise} \end{cases}$ $r_3(C_{2,q}) = q+1$ $r_3(C_{3,q}) = \begin{cases} (3q+1)/2 & \text{, if } q \text{ is odd} \\ (3q+2)/2 & \text{, otherwise} \end{cases}$	Dreyer [65]

2.3 Results

"This thing all things devours: Birds, beasts, trees, flowers; Gnaws iron, bites steel; Grinds hard stones to meal; Slays king, ruins town, And beats high mountain down." (Gollum)

— J.R.R. Tolkien, *The Hobbit*

In this section we will present our results concerning f-reversible processes. The complete proofs can be found in Appendix 5.1.5 and Appendix 5.1.5. Here we provide the main ideas of the proofs. The first results have been presented in Subsection 2.2.3, by Theorem 2.1, and Subsection 2.2.4, by Corollary 2.3. They concern the maximum length of the period on f-reversible processes, where it is proved that it is always at most 2. Next we summarize a tight upper bound for the maximum transient length, a characterization of the trees that need $n-3$ time steps to reach the periodic phase, and we prove the NP-completeness of f-Conversion Set for bipartite graphs with bounded degree.

2.3.1 A Tight Upper Bound on the Transient Length of f-Reversible Processes

An Equivalent Potential Function

First, we will prove that the potential function (2.5) is equivalent to

$$P'(t) = \sum_{v \in S_1(t)} (op_{t+1}(v) - f(v)) + \sum_{v \in S_2(t)} (f(v) - op_{t+1}(v)). \tag{2.6}$$

It is not so intuitive that Equation (2.5) and Equation (2.6) are equivalent. Table 2.3 shows the same analysis on Table 2.1 using the new potential function $P'(t)$. We can note that the found values are the same for $P(t)$ and $P'(t)$, at each time step t:

In order to prove the equivalence, we consider a partition of the edges whose ends have opposite states at time t as follows:

- $A(t) = \{(u, v) \in E(G) : \ u \in S_1(t), \ v \in S_1(t), \text{ and } c_t(u) \neq c_t(v)\}$;
- $B(t) = \{(u, v) \in E(G) : \ u \in S_2(t), \ v \in S_2(t), \text{ and } c_t(u) \neq c_t(v)\}$;
- $C(t) = \{(u, v) \in E(G) \setminus \{A(t) \cup B(t)\}, \text{ and } c_t(u) \neq c_t(v)\}$.

Note that each edge of $C(t)$ does not have both ends in the same set $S_1(t)$

Table 2.3: Sets $S_1(t)$, $S_2(t)$, values $op_{t+1}(v_i)$, $1 \leq i \leq 5$, and potential function $P'(t)$.

Config.	$S_1(t)$	$S_2(t)$	$op_{t+1}(v_1)$	$op_{t+1}(v_2)$	$op_{t+1}(v_3)$	$op_{t+1}(v_4)$	$op_{t+1}(v_5)$	$P'(t)$
c_0	$\{v_2, v_4, v_5\}$	$\{v_1, v_3\}$	0	1	1	1	1	3
c_1	$\{v_2, v_5\}$	$\{v_1, v_3, v_4\}$	1	1	0	0	0	5
c_2	$\{v_2, v_5\}$	$\{v_1, v_3, v_4\}$	0	1	1	1	1	5
c_3	$\{v_2, v_5\}$	$\{v_1, v_3, v_4\}$	1	1	0	0	0	5

or $S_2(t)$. Hence, we get that

$$\sum_{v \in S_1(t)} op_t(v) = 2|A(t)| + |C(t)| \quad \text{and} \quad \sum_{v \in S_2(t)} op_t(v) = 2|B(t)| + |C(t)|.$$

Therefore,

$$\sum_{v \in S_1(t)} op_t(v) - \sum_{v \in S_2(t)} op_t(v) = 2(|A(t)| - |B(t)|). \tag{2.7}$$

Lemma 2.7 *For $t \geq 0$, $P(t) = P'(t)$.*

Proof. Observe that $P(t)$ and $P'(t)$ can be rewritten as

$$P(t) = \left(\sum_{v \in S_1(t)} op_t(v) - \sum_{v \in S_2(t)} op_t(v) \right) + \left(\sum_{v \in S_2(t)} f(v) - \sum_{v \in S_1(t)} f(v) \right),$$

$$P'(t) = \left(\sum_{v \in S_1(t)} op_{t+1}(v) - \sum_{v \in S_2(t)} op_{t+1}(v) \right) + \left(\sum_{v \in S_2(t)} f(v) - \sum_{v \in S_1(t)} f(v) \right).$$

Thus, it is enough to prove that

$$\sum_{v \in S_1(t)} op_t(v) - \sum_{v \in S_2(t)} op_t(v) = \sum_{v \in S_1(t)} op_{t+1}(v) - \sum_{v \in S_2(t)} op_{t+1}(v). \tag{2.8}$$

We can observe that both terms of Equation (2.8) are defined on the same sets, but referring to the number of neighbors with opposite states at sequential instants, to each vertex v. We also consider similar sets to $A(t)$, $B(t)$ and $C(t)$, but referring to edges whose ends have opposite states at time $t+1$:

- $A'(t) = \{(u, v) \in E(G) : \ u \in S_1(t), \ v \in S_1(t) \text{ and } c_{t+1}(u) \neq c_{t+1}(v)\}$;
- $B'(t) = \{(u, v) \in E(G) : \ u \in S_2(t), \ v \in S_2(t) \text{ and } c_{t+1}(u) \neq c_{t+1}(v)\}$;
- $C'(t) = \{(u, v) \in E(G) \setminus \{A'(t) \cup B'(t)\} \text{ and } c_{t+1}(u) \neq c_{t+1}(v)\}$.

Since all vertices in $S_1(t)$ change their states and all vertices in $S_2(t)$ do not, all edges of $A(t)$ appear in $A'(t)$. The same holds for all edges of $B(t)$ in $B'(t)$. Moreover, since $A(t)$ and $A'(t)$ are defined over edges whose ends are in same set $S_1(t)$, we get that $A(t) = A'(t)$. It is analogous for $B(t)$ and $B'(t)$ with respect to $S_2(t)$. Therefore,

$$\sum_{v \in S_1(t)} op_{t+1}(v) - \sum_{v \in S_2(t)} op_{t+1}(v) = 2(|A'(t)| - |B'(t)|). \tag{2.9}$$

Equation (2.7) and Equation (2.9) show that Equation (2.8) is true. □

Monotonicity of the Potential Function

The next lemma shows that the potential function is nondecreasing. The proof gives the necessary intuition to obtain the new upper bound on the transient length. Let $\Delta P(t)$ be the *potential variation* from t to $t+1$, i.e., $\Delta P(t) = P(t+1) - P(t)$. Now, we will show that the potential variation is not negative, for all $t \in \mathbb{N}$.

Lemma 2.8 *$P(t)$ is a nondecreasing function.*

Proof. By Lemma 2.7, we can rewrite the potential variation as $\Delta P(t) = P(t+1) - P'(t)$. Therefore,

$$\begin{aligned}\Delta P(t) = & \sum_{v \in S_1(t+1)} (op_{t+1}(v) - f(v)) + \sum_{v \in S_2(t+1)} (f(v) - op_{t+1}(v)) \\ & - \sum_{v \in S_1(t)} (op_{t+1}(v) - f(v)) - \sum_{v \in S_2(t)} (f(v) - op_{t+1}(v)).\end{aligned}$$

Now, we can observe the contribution of each vertex to $\Delta P(t)$:

- If $v \in S_1(t)$ and $v \in S_1(t+1)$: $op_{t+1}(v) - f(v) - op_{t+1}(v) + f(v) = 0$;
- If $v \in S_2(t)$ and $v \in S_2(t+1)$: $f(v) - op_{t+1}(v) - f(v) + op_{t+1}(v) = 0$;
- If $v \in S_1(t)$ and $v \in S_2(t+1)$:
  - $f(v) - op_{t+1}(v) - op_{t+1}(v) + f(v) = 2(f(v) - op_{t+1}(v)) > 0$, since $f(v) > op_{t+1}(v)$;
- If $v \in S_2(t)$ and $v \in S_1(t+1)$:
  - $op_{t+1}(v) - f(v) - f(v) + op_{t+1}(v) = 2(op_{t+1}(v) - f(v)) \geq 0$, since $f(v) \leq op_{t+1}(v)$.

Since in each case the contribution is not negative, the lemma follows. □

A New Upper Bound on the Maximum Transient Length

Now we present a tight upper bound on the transient length of f-reversible processes. Lemma 2.9 gives an upper bound based on the potential function and the cardinality of S_2 at $\tau(c_0) - 1$, the last time step of the transient phase.

Lemma 2.9 *For c_0 such that $\tau(c_0) > 0$, $\tau(c_0) \leq P(\tau(c_0) - 1) - |S_2(\tau(c_0) - 1)| + 1$. Moreover, this bound is attained.*

The proof of Lemma 2.9 can be found in Appendix 5.1.5. Next, let us consider $V(G)$ partitioned according to time step $\tau(c_0) - 1$ as follows:

- $X_1 = \{v \in V(G) : c_{\tau(c_0)-1}(v) = 0 \text{ and } v \in S_1(\tau(c_0) - 1)\}$;
- $X_2 = \{v \in V(G) : c_{\tau(c_0)-1}(v) = 0 \text{ and } v \in S_2(\tau(c_0) - 1)\}$;
- $Y_1 = \{v \in V(G) : c_{\tau(c_0)-1}(v) = 1 \text{ and } v \in S_1(\tau(c_0) - 1)\}$;
- $Y_2 = \{v \in V(G) : c_{\tau(c_0)-1}(v) = 1 \text{ and } v \in S_2(\tau(c_0) - 1)\}$.

Let us denote the minimum threshold value by $f_{\min}$. Moreover, let us denote $S_f = \sum_{v \in V(G)} f(v)$. Thus, by Equation (2.5) and Equation (2.7), it follows that

$$\begin{aligned} P(\tau(c_0) - 1) &= \sum_{v \in S_2(\tau(c_0)-1)} f(v) - \sum_{v \in S_1(\tau(c_0)-1)} f(v) + 2|[X_1, Y_1]| - 2|[X_2, Y_2]| \\ &= S_f - 2\left(\sum_{v \in S_1(\tau(c_0)-1)} f(v)\right) + 2|[X_1, Y_1]| - 2|[X_2, Y_2]|. \end{aligned} \tag{2.10}$$

Next, by Lemma 2.9 and Equation (2.10) we obtain a tight upper bound on $\tau(c_0)$, for all c_0. This bound depends on the threshold function and the number of edges and vertices of the input graph, as follows.

Theorem 2.10 *For c_0 such that $\tau(c_0) > 0$,*

$$\tau(c_0) \leq \begin{cases} S_f - (n + 2f_{\min} - 2) & \text{, if } X_1 = \emptyset \text{ or } Y_1 = \emptyset; \\ S_f - (n + 2f_{\min} - 1) & \text{, if } p(c_0) = 1,\ X_1 \neq \emptyset, \text{ and } Y_1 \neq \emptyset; \\ S_f + 2m - 3n + 1 - & \\ \quad \sum_{v \in S_1(\tau(c_0)-1)} (2f(v) - 3) & \text{, if } p(c_0) = 2,\ X_1 \neq \emptyset, \text{ and } Y_1 \neq \emptyset. \end{cases}$$

Proof. **Case 1:** $X_1 = \emptyset$ or $Y_1 = \emptyset$:
In this case we get that $[X_1, Y_1] = \emptyset$. Thus

$$P(\tau(c_0) - 1) \leq S_f - 2\left(\sum_{v \in S_1(\tau(c_0)-1)} f(v)\right) - 2|[X_2, Y_2]|$$

and, by Lemma 2.9, we have

$$\begin{aligned}\tau(c_0) \leq\ & S_f - 2\left(\sum_{v\in S_1(\tau(c_0)-1)} f(v)\right) - (n - |S_1(\tau(c_0)-1)|) + 1 - 2|[X_2, Y_2]| \\ =\ & S_f - 2\left(\sum_{v\in S_1(\tau(c_0)-1)} f(v)\right) + |S_1(\tau(c_0)-1)| - n + 1 - 2|[X_2, Y_2]| \\ =\ & S_f - (n-1) - \sum_{v\in S_1(\tau(c_0)-1)} (2f(v) - 1) - 2|[X_2, Y_2]|.\end{aligned} \tag{2.11}$$

Since $S_f - (n-1)$ is constant, $|S_1(\tau(c_0)-1)| \geq 1$, and $2f(v) - 1 > 0$ for all $v \in V(G)$, the limit obtained by Equation (2.11) is maximum when $[X_2, Y_2] = \emptyset$, and $S_1(\tau(c_0)-1) = \{v\}$, where $f(v) = f_{\min}$. Hence, we obtain

$$\tau(c_0) \ \leq\ S_f - (n-1) - (2f_{\min} - 1) \ =\ S_f - (n + 2f_{\min} - 2).$$

Case 2: $p(c_0) = 1$, $X_1 \neq \emptyset$, and $Y_1 \neq \emptyset$:

Since every $v \in S_1(\tau(c_0)-1)$ belongs to $S_2(\tau(c_0))$, we get that $|N_{Y_1}(v)| \leq f(v)-1$, for every $v \in X_1$, and $|N_{X_1}(u)| \leq f(u) - 1$, for every $u \in Y_1$. Therefore

$$2|[X_1, Y_1]| \ =\ \sum_{v\in X_1} |N_{Y_1}(v)| + \sum_{u\in Y_1} |N_{X_1}(u)| \ \leq\ \sum_{v\in S_1(\tau(c_0)-1)} f(v) - |S_1(\tau(c_0)-1)|.$$

Thus, we can rewrite Equation (2.10) as

$$\begin{aligned}P(\tau(c_0)-1) \leq\ & \sum_{v\in S_2(\tau(c_0)-1)} f(v) - \sum_{v\in S_1(\tau(c_0)-1)} f(v) + \sum_{v\in S_1(\tau(c_0)-1)} f(v) - \\ & |S_1(\tau(c_0)-1)| - 2|[X_2, Y_2]| \\ \leq\ & \sum_{v\in S_2(\tau(c_0)-1)} f(v) - |S_1(\tau(c_0)-1)|.\end{aligned}$$

By Lemma 2.9 and the fact that $|S_1(\tau(c_0)-1)| \geq 2$, it follows that

$$\begin{aligned}\tau(c_0) \leq\ & \sum_{v\in S_2(\tau(c_0)-1)} f(v) - |S_1(\tau(c_0)-1)| - |S_2(\tau(c_0)-1)| + 1 \\ =\ & \left(\sum_{v\in V(G)} f(v) - \sum_{v\in S_1(\tau(c_0)-1)} f(v)\right) - n + 1 \\ \leq\ & S_f - (n + 2f_{\min} - 1).\end{aligned}$$

Case 3: $p(c_0) = 2$, $X_1 \neq \emptyset$, and $Y_1 \neq \emptyset$:

As in Case 2, $P(\tau(c_0)-1)$ is maximum when $[X_2, Y_2] = \emptyset$, while $|[X_1, Y_1]| \leq$

$m - |S_2(\tau(c_0)) - 1|$. It means that $S_1(\tau(c_0)-1)$ and $S_2(\tau(c_0)-1)$ induce independent sets. Furthermore, $d(v) = 1$ for all $v \in S_2(\tau(c_0) - 1)$. Hence, we can rewrite Equation (2.10) as

$$\begin{aligned} P(\tau(c_0)-1) \leq\ & S_f - 2\left(\sum_{v \in S_1(\tau(c_0)-1)} f(v)\right) + 2(m - |S_2(\tau(c_0)-1)|) \\ =\ & S_f + 2m - \ 2\left(\sum_{v \in S_1(\tau(c_0)-1)} f(v)\right) - 2|S_2(\tau(c_0)-1)|. \end{aligned}$$

Hence, by Lemma 2.9 we complete the proof:

$$\begin{aligned} \tau(c_0) \leq\ & S_f + 2m - 2\left(\sum_{v \in S_1(\tau(c_0)-1)} f(v)\right) - 3|S_2(\tau(c_0)-1)| + 1 \\ =\ & S_f + 2m + 1 - 2\left(\sum_{v \in S_1(\tau(c_0)-1)} f(v)\right) - 3(n - |S_1(\tau(c_0)-1)|) \\ \leq\ & S_f + 2m - 3n + 1 - \sum_{v \in S_1(\tau(c_0)-1)} (2f(v) - 3) \\ =\ & S_f + 2m - 3n + 1 - \sum_{v \in S_1(\tau(c_0)-1)} (2f(v) - 3). \end{aligned}$$

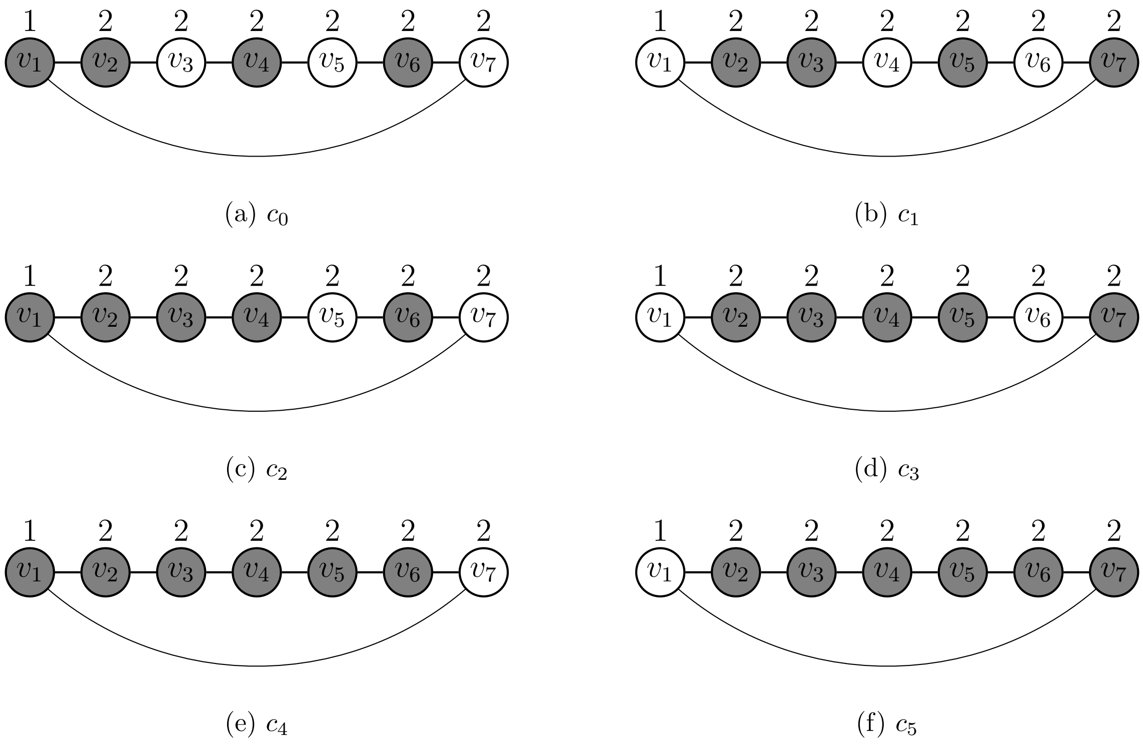
□

Figure 2.13 shows an example illustrating that the bound on Case 1 of Theorem 2.10 is tight. We consider an f-reversible process $R_f(C_n, c_0)$ on odd cycles $C_n = v_1v_2\ldots v_n$, in which $f(v_1) = 1$, and $f(v_i) = 2$, for all $i \neq 1$. Moreover $c_0(v_1) = 0$ if and only if $i > 1$ and i is odd. This process is uplifting and satisfies the following conditions:

- $v_1 \in S_1(t)$, for all $t < \tau(c_0)$;
- For every $j > 1, v_j \in S_1(t)$, for all $t < (j-2)$;
- For every $j > 1, v_j \in S_2(t)$, for all $t \geq (j-2)$.

Thus $\tau(c_0) = n - 1$, as well as in Case 1 of Theorem 2.10. However, such a bound can be arbitrarily far from $\tau(c_0)$. For instance, consider an f-reversible process on a star G with n vertices and whose central vertex is v. Suppose $f(v) = n$ and $f(u) = 1$, for all $u \neq v$. If $c_0(v) = 1 - c_0(u)$, for every $u \neq v$, then $\tau(c_0) = 1$, although Theorem 2.10 results on $\tau(c_0) \leq (n-1)$. Notice that Lemma 2.9 gives to us the correct value.

Figure 2.14e is an example attaining the bound on Case 2 of Theorem 2.10, where every vertex has threshold value 2. With respect to Case 3, we can cite an 1-reversible process on a path $P_n = \{v_1, v_2, \ldots, v_n\}$ with $n \geq 3$, in which v_1 has the



(a) c_0 (b) c_1 (c) c_2 (d) c_3 (e) c_4 (f) c_5

Figure 2.13: Transient phase of $R_f(C_7, c_0)$.

opposite state from the all others. Such a process has transient length equal to $n-2$, as well as in Case 3.

2.3.2 The Transient Length of 2-Reversible Processes on Trees

In this subsection we consider the maximum transient length of 2-reversible processes on trees. If $n \leq 4$ and $\tau(c_0) > 0$, it can be seen that $\tau(c_0) \leq n-2$, for all c_0. Such a limit holds when T is a path with 3 vertices, whose central vertex has the opposite state to the others Figure 2.14a. However, all non-periodic configurations with four vertices Figure 2.14 have transient length equal to $n-3$. Actually, we prove that $n-3$ is a tight upper bound for all initial configurations on trees with $n \geq 4$. If $n \leq 2$ then all configurations are periodic. In fact, the proof gives a characterization of all trees requiring $n-3$ time steps, with $n \geq 4$.

Theorem 2.11 *For T a tree with $n \geq 4$, $\tau_2(T) \leq n-3$.*

Proof. The theorem follows directly in both Cases 2 and 3 of Theorem 2.10, while Case 1 shows that $\tau_2(T) \leq n-2$. For trees T with $f(v) = 2$ for all $v \in V(T)$,

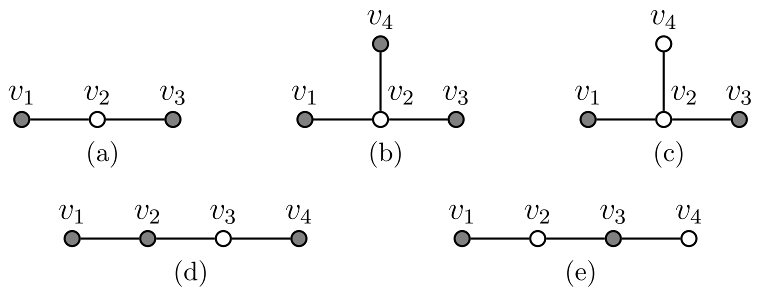


Figure 2.14: All non-periodic configurations with three and four vertices.

we resort to Equation (2.10) and Lemma 2.9 as follows, where $p = |S_1(\tau(c_0) - 1)|$:

$$\begin{aligned}\tau(c_0) &\leq S_f - 2\left(\sum_{v\in S_1(\tau(c_0)-1)} f(v)\right) + 2|[X_1, Y_1]| - 2|[X_2, Y_2]| - \\ &\qquad |S_2(\tau(c_0) - 1)| + 1 \qquad (2.12)\\ &\leq 2(n) - 2(2p) - (n-p) + 1 + 2|[X_1, Y_1]| - 2|[X_2, Y_2]| \\ &= n - 3p + 1 + 2|[X_1, Y_1]| - 2|[X_2, Y_2]|.\end{aligned}$$

We split the proof into cases according to the cardinalities of sets X_1 and Y_1.

Case 1: $X_1 = \emptyset$ or $Y_1 = \emptyset$:

By Equation (2.12) and the fact that $p \geq 1$, if $p > 1$ or $[X_2, Y_2] \neq \emptyset$ then $\tau(c_0) \leq n - 4$. Therefore, let us suppose that $[X_2, Y_2] = \emptyset$ and $S_1(\tau(c_0) - 1) = \{v\}$, where $N(v) = \{v_1, v_2, \ldots, v_{d(v)}\}$, $d(v) \geq 2$. Moreover, let us consider v being the root of T and $c_{\tau(c_0)-1}(v) = 0$, without loss of generality. Thus, each subtree T_{v_i}, rooted at v_i, has all its vertices with the same state at $\tau(c_0)-1$, for each $i \in \{1, 2, \ldots, d(v)\}$. Since only v must change its state at time $\tau(c_0) - 1$, there must exist at least two subtrees T_{v_i} whose vertices have state 1 at time $\tau(c_0)-1$. Furthermore there exists at most one subtree with all its vertices having state 0 at time $\tau(c_0)-1$. Hence $p(c_0) = 1$ in this case.

Let us consider that T_{v_1} is the last subtree whose vertices reach their final state. Since the process follows concurrently in all subtrees, we split the analysis in two sub-cases based on the degree of v.

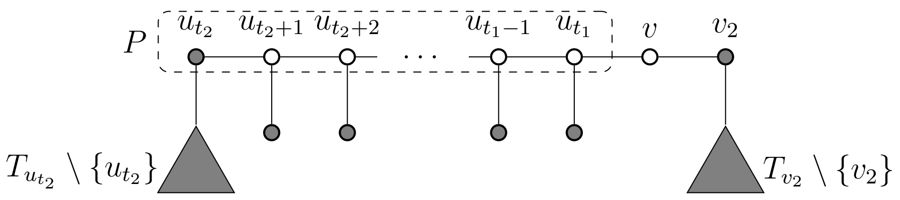


Figure 2.15: Representation of configuration c_{t_2} in Sub-case 1.1.

Sub-case 1.1: $d(v) = 2$.

Notice that all vertices in $V(T)\setminus\{v\}$ must have the same state 1 at time $\tau(c_0)-1$, where the process is uplifting from the leaves toward v.

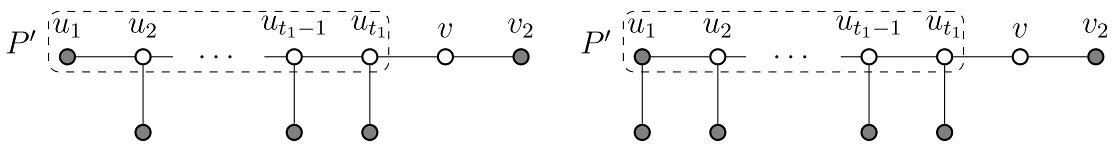
Let t_1 and t_2 be the time steps in which T_{v_1} and T_{v_2} are uplifting, respectively. Since $t_1 = \tau(c_0) - 1$, we get that both of v and v_1 must keep state 0 from t_2 to t_1. If $t_1 = t_2$, then t_1 is maximum when $V(T_{v_1})$ induces a maximum path that is uplifting. Thus T_{v_1} is a path whose states of the vertices alternate. Furthermore T_{v_2} must have the same length as T_{v_1}. Thus T is an odd size path whose vertices alternate, implying that $\tau(c_0) = \frac{n-1}{2}$. Hence, if $\tau(c_0) = n - 2$ then $n = 3$, and if $\tau(c_0) = n - 3$ then $n = 5$. These cases are depicted in Figure 2.14a and Figure 2.17b, respectively.

On the other hand, if $t_1 > t_2$, it means that v_1 must keep state 0 from t_2 to $t_1 - 1$. Thus $t_1 - t_2$ is equal to the length of a path $P = \{u_{t_2}, u_{t_2+1}, \ldots, u_{t_1}\}$, where $u_{t_1} = v_1$ and each u_i reaches state 1 at time i, for all $i \in \{t_2, t_2 + 1, \ldots, t_1\}$. Moreover, for every u_i we get that $d(u_i) > 2$ and each one must have at most one neighbor with opposite state, from time t_2 to time i. The configuration obtained at time t_2 which maximizes $|P|$ is depicted in Figure 2.15, where all vertices have state 1, unless v and those of $V(P) \setminus \{u_{t_2}\}$.

For every tree T, we can obtain a tree T' such that $\tau_2(T') \geq \tau_2(T)$ as follows. To each pair of vertices w and w' of $T_{v_2} \setminus \{v_2\}$, remove them from T_{v_2}, add w to P and, add an edge between w and w'. Moreover, assign $c_0(w) = 0$ and $c_0(v_2) = c_0(w') = 1$. If $|T_{v_2}|$ is even then there remain two vertices in T_{v_2}, by the previous procedure. Thus, remove the last neighbor w'' of v_2 from T_{v_2} and add it to T_{v_1} as a neighbor of the vertex of P at maximum distance from v, such that $c_0(w'') = 1$. Figure 2.16a and Figure 2.16b represent the cases in which $|T_{v_2}|$ is odd or even, respectively. Thus path P increases, yielding a new path $P' = \{u_1, u_2, \ldots, u_{t_1}\}$. Hence it is obtained one more time for each pair of moved vertices.

Note that $\tau_2(c_0) = |P'|$. Hence, if $|T_{v_2}|$ is odd then $\tau(c_0) = |P'| \leq \frac{n-1}{2}$ and it follows that:

- $\frac{n-1}{2} = n - 2 \;\Rightarrow\; n = 3$ and $|P'| = 1$ (Figure 2.14a);
- $\frac{n-1}{2} = n - 3 \;\Rightarrow\; n = 5$ and $|P'| = 2$ (Figure 2.17a).



(a) T' when $|T_{v_2}|$ is odd. (b) T' when $|T_{v_2}|$ is even.

Figure 2.16: Representation of tree T' obtained from a tree T, such that $\tau_2(T') \geq \tau_2(T)$.

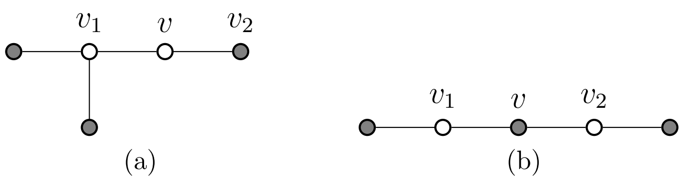


Figure 2.17: All configurations with $\tau(c_0) \geq n-3$, such that $X_1 = Y_1 = \emptyset$, $d(v) = 2$ and $n \geq 5$.

Finally, if $|T_{v_2}|$ is even then $\tau(c_0) = |P'| = \frac{n-2}{2}$ and it follows that:

- $\frac{n-2}{2} = n-2 \;\Rightarrow\; n = 2$ and $|P'| = 0$;
- $\frac{n-2}{2} = n-3 \;\Rightarrow\; n = 4$ and $|P'| = 1$ (Figure 2.14d).

Sub-case 1.2: $d(v) \geq 3$.

Let us suppose that all non-leaf vertices reach their final states exactly one time step after all of their children. Let $P = \{p_1, p_2, \ldots, p_\ell\}$ be a longest path from $p_1 = v$ until a leaf p_ℓ. Thus, for each internal vertex p_i, we get that p_i reaches its final state exactly after p_{i+1}, $i < \ell$ and $\tau(c_0) = |P|$, where $|P| \leq n-3$. Figure 2.18 depicts this case, in which $d(v) = 3$ and the internal vertices of P must alternate their states, since they have degree equal to 2. Therefore v must have 2 neighbors with opposite states, for all $t < \tau(c_0) - 1$. If $d(v) \geq 4$ or $d(p_i) \geq 3$ for some internal vertex p_i then $\tau(c_0) \leq n-4$, in this case.

Now, let $u \neq v$ be a non-leaf vertex whose state does not change even if all of its children reach their final states. Moreover, let consider u the farther vertex from v satisfying this property. Since all leaves of a subtree T_{v_i} must have the same state, say $s \in \{0,1\}$, we have $d(u) = 2$. Moreover u and its parent w must have the same initial state 0, when the child of u achieves its final state, and u changes its state exactly one time step after w. In other words, the uplifting of T_{v_i} follows from the leaves to v, but it stops at u, which is "released" by w.

Therefore, either w is released by at least two children, or w also depends on its parent. Hence, there exists a path $W = \{w_x, w_{x-1}, \ldots, w_1\}$, such that w_i depends on its parent w_{i-1} to change its state, for all $i \in \{1, \ldots, x-1\}$. Thus it follows

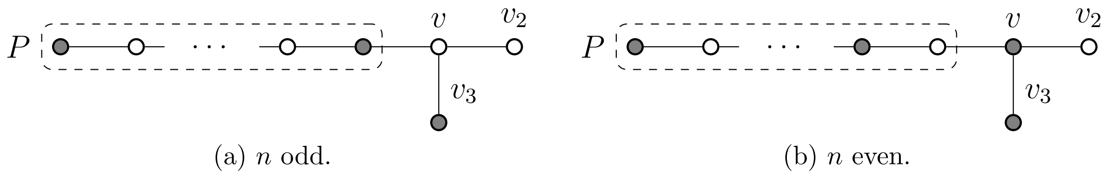


Figure 2.18: Representation of the initial configurations of trees in which $\tau_2(c_0) = n-3$ and all non-leaf vertices reach their final states exactly one time step after all of their children.

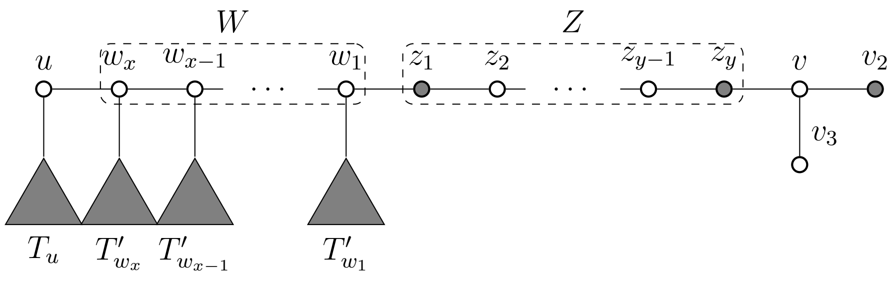


Figure 2.19: Representation of tree T, where $v \in S_1(t)$, for all $t \leq \tau(c_0) - 1$.

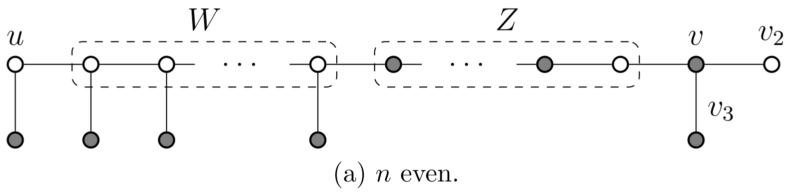
that $d(w_i) \geq 3$, for all $w_i \in W$. Moreover w_1 is the first vertex of W to be released, where the vertices in $W \cup \{u\}$ change their initial states from w_1 to u. If $z_1 = N(w_1) \cap P$ does not reach its final state before w_1, the process follows changing the states of the vertices from w_1 to u and returns to w_1. Otherwise, the process takes fewer time steps, since it would follow simultaneously to $T \setminus T_{z_1}$. In this case, all vertices on path $Z = \{z_1, z_2, \ldots, z_y\}$ must alternate their states until T_{w_1} is uplifting.

Let $T'(w_i)$ be the subtree of w_i which does not intersect P, for all $i \in \{1, \ldots, x\}$. To maximize the transient length, w_i must keep state 0 from $t = 0$ until its parent and all of its children achieve state 1. Thus w_i waits until T'_{w_i} is uplifting (see Figure 2.19). Moreover, since w_i does not affect the state of its children, the maximum number of time steps required to the uplifting of T'_{w_i} is $|T'_{w_i}|-1$. Thus, it follows that

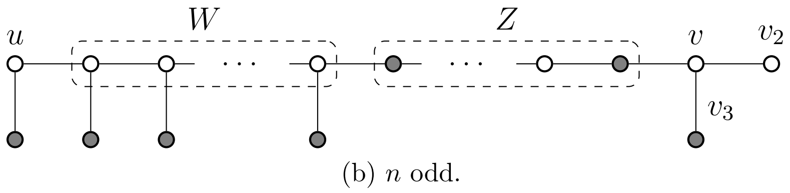
$$\begin{aligned}
\tau(c_0) &\leq 2x + y + 2 + \sum_{i=1}^{x} \tau_2(T'_{w_i}) + \tau_2(T_u) \\
&\leq 2x + y + 2 + \sum_{i=1}^{x} (|T'_{w_i}| - 1) + (|T_u| - 1) \\
&= x + y + 1 + \sum_{i=1}^{x} |T'_{w_i}| + |T_u| \\
&\leq n - 3.
\end{aligned}$$

Thus $\tau(c_0) = n - 3$ when T is as in Figure 2.20a and Figure 2.20b, where all subtrees T'_{w_i} and T_u have exactly one vertex and all vertices of Y have degree 2.

Note that if $x = 0$ then the obtained configurations are equivalent to those in Figure 2.18. On the other hand, if n is even and $y = 0$ then v "starts" the process, where its state must be equal to the states of the leaves of P, as well as when n is odd and $y = 0$, but with initial state opposite to that of the leaves. In the last case it is possible to keep the state of v, adding one neighbor to v with opposite state, yielding a tree with even number of vertices Figure 2.21a. The effect is to extend W including v, increasing the transient length by one time step and the number of vertices also by one, keeping the upper bound on $n - 3$. On the other hand, no other vertex can be added as a neighbor of v, since v already have two neighbors



(a) n even.



(b) n odd.

Figure 2.20: Representation of a general initial configuration of trees in which $\tau(c_0) = n - 3$ and at least one non-leaf vertex reaches its final state after all of its children.

of each state. Finally, we can obtain the configuration given in Figure 2.21b, where the additional vertex is added as a neighbor of v_2, in which the effect is the same as the previous case. Moreover, any additional vertex does not increase the transient length. Hence, no other initial configuration is possible.

Case 2: $X_1 \neq \emptyset$ and $Y_1 \neq \emptyset$:

Since $|[X_1, Y_1]| \leq p-1$ and $V(S_1(\tau(c_0) - 1))$ induces a forest, by Equation (2.12) it follows that

$$\begin{aligned}\tau(c_0) \leq\ & n - 3p + 1 + 2(p - 1) - 2|[X_2, Y_2]| \\ &= n - p - 1 - 2|[X_2, Y_2]|.\end{aligned}$$

Therefore, if $p > 2$ or $[X_2, Y_2] \neq \emptyset$ then $\tau(c_0) \leq n - 4$ and the theorem follows. Thus, let us suppose $[X_2, Y_2] = \emptyset$ and consider $S_1(\tau(c_0) - 1) = \{u, v\}$, such that $(u, v) \in E(T)$ and $c_{\tau(c_0)-1}(v) = 1 - c_{\tau(c_0)-1}(u)$.

Since every $w \in S_2(\tau(c_0) - 1)$ has at most one neighbor with opposite state (u or v), w belongs to $S_2(\tau(c_0))$. Thus $|S_1(\tau(c_0))| = 2$ only if $S_1(\tau(c_0)) = \{u, v\}$, implying that $c_{\tau(c_0)-1}$ is periodic, a contradiction. Therefore, either $p(c_0) = 1$, where u and v have opposite states, or $S_1(\tau(c_0))$ contains exactly one vertex, which is either u or v.

Suppose $p(c_0) = 2$, where $S_1(\tau(c_0)) = v$ and $c_{\tau(c_0)-1}(v) = i$, $i \in \{0, 1\}$.

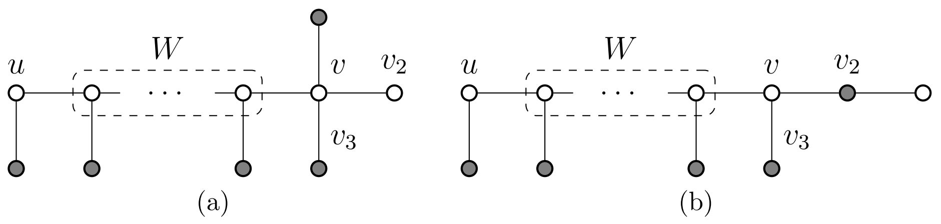


Figure 2.21: Representation of trees T with $\tau(c_0) = n - 3$, in which $y = 0$ and n even.

Thus $d(v) \geq 4$ and v has at least one neighbor with state i and at least three with state $1-i$ (where u is one of them), at $\tau(c_0)-1$. Since $(N(u) \setminus \{v\}) \subset S_2(\tau(c_0)-1)$, it follows that the value of $\tau(c_0)$ is at most the maximum length of a path P whose vertices alternate their states, where P is a subtree of u. Hence $\tau(c_0) \leq n - |N(v) \cup \{v\}| = n-5$.

Now, suppose that $p(c_0) = 1$. Thus $c_{\tau(c_0)-1}(w) = 1 - c_{\tau(c_0)-1}(u)$, for all $w \in N(u)$, and $c_{\tau(c_0)-1}(w') = 1 - c_{\tau(c_0)-1}(v)$, for all $w' \in N(v)$. Therefore $\tau(c_0)$ is maximum when a subtree T'_u of u has maximum transient length. Thus T'_u is a path whose vertices alternate their states. Hence, we get that $\tau(c_0) \leq n - |N(v) \cup \{v\}| \leq n-3$. Moreover, the previous limit is obtained only if $d(u) = d(v) = 2$, where $\tau(c_0) = 1$, since $v \in S_2(1)$. This situation is depicted in Figure 2.14e. Finally, consider $d(u) \geq 3$. Since $V(G) \setminus \{u, v\} = S_2(\tau(c_0)-1)$, all subtrees of u which do not contain v and every subtree of v that do not contain u must reach their final states at same time step $\tau(c_0)-1$. Analogously, $\tau(c_0)$ is maximum when a subtree T'_v of v is a path whose vertices alternate their states. Hence $\tau(c_0) \leq n - |N(u) \cup \{u, v\}| = n-4$. $\square$

Let $\#\tau_f(G, q)$ be the number of configurations c_0 such that $\tau(c_0) = q$ for an f-reversible process on G. Thus, we prove that $\#\tau_2(T, n-3) = O(n)$ for trees T with $n \geq 4$.

Corollary 2.12 *For T a tree with $n \geq 4$,*

$$\#\tau_2(T, n-3) = \begin{cases} 4 & , \text{ if } \ n = 4; \\ 3 & , \text{ if } \ n = 5; \\ \frac{n}{2} & , \text{ if } n \geq 6 \text{ and } n \text{ is even}; \\ \frac{n-3}{2} & , \text{ if } \ n \geq 7 \text{ and } n \text{ is odd}. \end{cases}$$

It follows also from Theorem 2.11 an algorithm that generates all such initial configurations.

Corollary 2.13 *For T a tree with $n \geq 4$, $\tau_2(T) = n-3$ if and only if c_0 is output by Algorithm 1.*

Both proofs of Corollary 2.12 and Corollary 2.13 can be found in Appendix 5.1.5.

2.3.3 NP-Completeness of f-Conversion Set

In this section, we prove the NP-completeness for bipartite graphs of f-Conversion Set. The proof is a reduction through a restriction of 3SAT, where each clause has 2 or 3 literals and each variable occurs in at most 3 clauses [75]. It can be polynomially solved if all clauses have exactly 2 or exactly 3 literals [118].

Algorithm 1: Generate all trees T with $n \geq 4$ and corresponding initial configurations leading to $\tau_2(T) = n - 3$.

```
Input: Number $n$ of vertices.
Output: Trees $T_j$, each with the corresponding initial configuration $C_j$ such that
        $\tau(C_j) = n - 3$.
$V \leftarrow \{v_1, v_2, \ldots, v_n\}$;
$E \leftarrow \emptyset$;
$j \leftarrow 1$;
for $i \leftarrow 1$ to $n - 2$ do                                  // Lines 4 -- 7 result on tree $T_1$.
    $E \leftarrow E \cup (v_i, v_{i+1})$;
    if $i = n - 2$ then
        $E \leftarrow E \cup (v_i, v_{i+2})$;
for $i \leftarrow 1$ to $n$ do                                      // Lines 8 -- 12 result on configuration $C_1$.
    if $i$ is odd then
        $c_0(v_i) \leftarrow 1$;
    else
        $c_0(v_i) \leftarrow 0$;
$T_j \leftarrow G(V, E)$;
$C_j \leftarrow c_0$;        // Fig. 2.18a and Fig. 2.18b are obtained for $n$ odd and even, respectively.
$j \leftarrow j + 1$;
if $n = 4$ or $n = 5$ then
    $c_0(v_4) \leftarrow 1 - c_0(v_4)$; // Lines 17 -- 21 result on Fig. 2.14b and Fig. 2.17a for $n = 4$ and $n = 5$, respectively.
    if $n = 5$ then
        $c_0(v_3) \leftarrow 1 - c_0(v_3)$;
    $T_j \leftarrow G(V, E)$;
    $C_j \leftarrow c_0$;
    $j \leftarrow j + 1$;
    $E \leftarrow E \setminus (v_n, v_{n-2})$; // Lines 23 -- 29 result on Fig. 2.14d and Fig. 2.17b for $n = 4$ and $n = 5$, respectively.
    $E \leftarrow E \cup (v_{n-1}, v_n)$;
    if $n = 5$ then
        $c_0(v_3) \leftarrow c_0(v_4)$;
        $c_0(v_4) \leftarrow 1 - c_0(v_4)$;
    $T_j \leftarrow G(V, E)$;
    $C_j \leftarrow c_0$;
    if $n = 4$ then          // Lines 31 -- 34 obtain the tree and configuration given by Fig. 2.14e.
        $j \leftarrow j + 1$;
        $c_0(v_4) \leftarrow 1 - c_0(v_4)$;
        $T_j \leftarrow G(V, E)$;
        $C_j \leftarrow c_0$;
else
    for $i \leftarrow 3$ to $n - 3$ do  // Lines 36 -- 42 obtain Fig. 2.20a and Fig. 2.20b for $n$ even and odd, respectively.
        if $i$ is odd then                   // For $n$ even and $i = n - 3$, it is obtained Fig. 2.21a.
            $E \leftarrow E \setminus (v_i, v_{i-1})$;
            $E \leftarrow E \cup (v_{i-1}, v_{i+1})$;
            $T_j \leftarrow G(V, E)$;
            $C_j \leftarrow c_0$;
            $j \leftarrow j + 1$;
        if $i = n - 3$ and $n$ is even then          // Lines 43 -- 48 obtain Fig. 2.21b for $n$ even.
            $E \leftarrow E \setminus (v_{i+1}, v_{i+3})$;
            $E \leftarrow E \cup (v_i, v_{i+3})$;
            $T_j \leftarrow G(V, E)$;
            $C_j \leftarrow c_0$;
            $j \leftarrow j + 1$;
```

Given an integer $q > 0$, Dourado *et al.* [59] proved that determining if $r_2(G) \leq q$ is NP-hard. We say that two disjoint vertex subsets A and B are a *bad pair* of an f-reversible process $R_f(G, c_0)$ if:

- each vertex of A has the opposite state from the vertices of B;
- $|N_B(v)| \geq f(v)$, for all $v \in A$;
- $|N_A(u)| \geq f(u)$, for all $u \in B$.

As proved by Dourado *et al.* [59], every uplifting f-reversible process cannot contain any bad pair.

Lemma 2.14 *If $R_f(G, c_0)$ is uplifting and, for some $v \in V(G)$, $f(v) = d(v)$ and $f(u) = 1$ for every $u \in N(v)$, then $c_t(v) = 1$ and $c_t(u) = 1$ for some $u \in N(v)$ and every $t \geq 0$.*

Proof. Suppose by contradiction that the lemma is false. Since $R_f(G, c_0)$ is uplifting, if there exists a time step $t_0 \geq 0$ such that $c_{t_0}(v) = 0$ then there exists a time step $t > t_0$ in which v changes its state. Suppose t to be the smallest. In this case, $c_{t-1}(v) = 0$ and $c_{t-1}(u) = 1$ for all $u \in N(v)$. Hence, sets $A = \{v\}$ and $B = N(v)$ are a bad pair of $R_f(G, c_0)$, a contradiction. The argument for the existence of a vertex $u \in N(v)$ which has state 1 through the process is similar. □

Theorem 2.15 *f-Conversion Set is NP-complete for G bipartite with maximum degree 3 and* $\mathrm{Im}_f = \{1, 2, 3\}$.

Proof. f-Conversion Set is in NP because, given an integer $q > 0$, we can execute the process from c_0, that has q vertices with state 1, to $c_{\tau(c_0)}$. Moreover, we can obtain c_{t+1} from c_t in $O(n+m)$ for all $t \geq 0$, because by Case 1 of Theorem 2.10 we can execute the whole process in $O((n+m)(S_f - n))$.

Let F be an instance of the restricted 3SAT problem with n variables $X_1, X_2, \ldots, X_n$ and m clauses $C_1, C_2, \ldots, C_m$. We construct a graph G of f-Conversion Set as follows. For each variable X_i, G contains three vertices denoted by x_i, $\overline{x_i}$ and a_i, $1 \leq i \leq n$. For each clause C_j, there is one vertex denoted by c_j, $1 \leq j \leq m$. Each vertex a_i is adjacent only to x_i and $\overline{x_i}$, $1 \leq i \leq n$. Moreover, edges $x^i c_j$, $x^i \in \{x_i, \overline{x_i}\}$, are added if and only if $x^i \in C_j$ in F, for all $1 \leq i \leq n$ and $1 \leq j \leq m$. Figure 2.22 depicts the graph, initial configuration and threshold values obtained from $F = (X_1 \vee \overline{X_2}) \wedge (\overline{X_1} \vee X_2 \vee X_3) \wedge (X_2 \vee \overline{X_3} \vee X_4) \wedge (\overline{X_3} \vee \overline{X_4})$.

Let $A = \bigcup a_i$, $X = \bigcup \{x_i \cup \overline{x_i}\}$ and, $C = \bigcup c_i$ denote the sets of vertices called *auxiliar*, *literal* and, *clause* vertices, respectively. Note that each vertex of A has degree 2 and, since each clause has 2 or 3 literals and each literal occurs in 1 or 2 clauses, the maximum degree of G is 3. Moreover, $A \cup C$ and X are a bipartition of $V(G)$, where each part induces an independent set. Therefore G is a bipartite

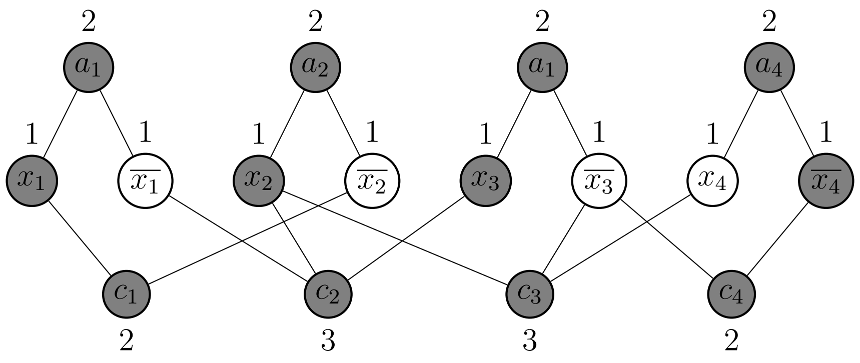


Figure 2.22: A labelling with $r_f(G)$ vertices satisfying $F = (X_1 \vee \overline{X_2}) \wedge (\overline{X_1} \vee X_2 \vee X_3) \wedge (X_2 \vee \overline{X_3} \vee X_4) \wedge (\overline{X_3} \vee \overline{X_4})$.

graph with maximum degree 3. To complete the transformation, we assign $f(x) = 1$, for each $x \in X$, and $f(v) = d(v)$ for all $v \in V(G) \setminus \{X\}$.

Now, we prove that F is satisfying if and only if $r_f(G) = m+2n$. By Lemma 2.14, we get that all vertices $v \in A \cup C$ and at least one of x_i or $\overline{x_i}$ must have state 1 in c_0, for all $1 \leq i \leq n$. Thus $r_f(G) \geq m + 2n$. We can consider that $c_0(x^i) = 1$ if and only if x^i has its true value in F, $x^i \in \{x_i, \overline{x_i}\}$.

If F is satisfying then for each vertex c_j, $1 \leq j \leq m$, there is at least a neighbor that is a literal vertex with initial state 1. Furthermore, for each pair of literal vertices x_i and $\overline{x_i}$, $1 \leq i \leq n$, only one of them has its initial state equal to 1. Thus, clearly $c_1(v) = 1$ for every $v \in V(G)$ and $r_f(G) = m + 2n$. Now, if $r_f(G) = m + 2n$ then, for each pair of vertices x_i and $\overline{x_i}$, only one of them must have its initial state equal to 1, because all vertices $u \in A \cup C$ have their initial state equal to 1. Moreover, since the process is uplifting, each clause vertex c_j, $1 \leq j \leq m$, is adjacent to a literal vertex with initial state 1, by Lemma 2.14. Hence, all clauses are satisfied, concluding the proof. □

We also analyze the complexity of finding a minimum f-conversion set on general graphs when $f(v) = d(v)$, for all $v \in V(G)$. In this case, if an edge is such that the states of its ends are the same then these vertices will never change their states, showing that $r_f(G) \geq \beta(G)$. We establish the following result relating $r_f(G)$ and $\beta(G)$ in this situation.

Theorem 2.16 *Let $R_f(G, c_0)$ be such that $f(v) = d(v)$ for all $v \in V(G)$. If a minimum vertex cover of G exists that is not an independent set then $r_f(G) = \beta(G)$. Otherwise, $r_f(G) = \beta(G) + 1$.*

Proof. Let C be a minimum vertex cover of G and let us consider that a vertex v belongs to C if and only if $c_0(v) = 1$. Clearly $r_f(G) \geq \beta(G)$, otherwise, there must exist at least one edge whose ends have state zero in c_0, meaning that the process does not uplift. Let I_t be the set of vertices v such that $c_t(v) = 0$, for all $0 \leq t \leq \tau(c_0) - 1$. Hence, I_t must induce an independent set in G and $r_f(G)$ is obtained taking I_0 the greatest possible.

Suppose that C does not induce an independent set in G. Thus there is at least one edge $e = (u, v)$ in $G[C]$ and, hence, u and v remain with state 1 forever. Suppose by contradiction that the process is not uplifting. Let us also suppose that there exists an edge e' whose ends have state zero in $c_{\tau(c_0)}$. Furthermore, let $t > 0$ be the first time step in which such an edge e' appears. Since at time $t - 1$ there were not any such edges, all vertices $z \in I_{t-1}$ change their states. Thus, I_t induces an independent set, a contradiction. Hence, if $p(c_0) = 1$, then the process is uplifting.

Thus, suppose that $p(c_0) = 2$. In this case, there must exist a bad pair A and B. Hence A and B induce independent sets, and since no edge whose ends have state zero is formed through the process, we get that $\tau(c_0) = 0$ and $G[A \cup B]$ is a connected component which does not contain u and v, a contradiction.

Finally, suppose that any minimum vertex cover C of G induces an independent set. Thus, we need more vertices than $\beta(G)$ for the uplifting. It is enough to show that we need only one vertex more than a minimum covering. Let $C \cup \{v'\}$ be the vertex subset of G with initial state 1, for some $v' \in V(G) \setminus C$. Hence, since G is connected, there is an edge in $G[C \cup \{v'\}]$ and the same argument of the previous case works now, completing the proof. □

Corollary 2.17 *If G is not bipartite and $f(v) = d(v)$ for all $v \in G$ then $r_f(G) = \beta(G)$.*

Proof. Since there is no minimum vertex cover of G inducing an independent set, the corollary follows from Theorem 2.16. □

Chapter 3

Some Irreversible Processes with Thresholds

"Every time we deal with an enemy, we create two more." (Tyrion Lannister)

— G. Martin, *A Game of Thrones*

In this chapter we will present some irreversible processes on graphs related to some invariants of distributed computing. The first one concerns on the study of the *AND-OR model*, which describes a deadlock prevention model. In this model each vertex v has a label among two, which describes the amount of out-neighbors of v required to be realized in order to avoid a wait state of v. In particular we show that some classical convexity parameters are closed related to stable properties of the AND-OR model.

The another irreversible process is a generalization of processes where the vertices depend on certain amount of neighbors to be added in the initial vertex set S taken as input. In particular, each vertex u has a list of subset of $N_G(u)$, such that u is included in S if some such a subset is in S. We consider two versions, where is required that an exact subset of $N_G(u)$ of the initial list is in S, and where just a subset of $N_G(u)$ of the initial list is in S.

3.1 Threshold Process Based on the AND-OR Model

As described by Barbosa and Benevides [14], a deadlock situation can be characterized by the permanent impossibility for a group of processes to progress with their tasks due to the occurrence of a condition that prevents at least one needed resource from being granted to each of the processes in that group. Note that a *necessary* condition for the existence of a deadlock in the distributed computation is the existence of cycles of dependency.

A useful abstraction to analyze deadlock situations is the wait-for graph $G = (V, E)$, where E is a set of directed edges, and an edge exists in E directed away from $v_i \in V$ towards $v_j \in V$ if v_i is blocked for some condition that v_j may relieve. The graph G changes dynamically as the computation progresses, and what determines the evolution of G by allowing for changes in the set of its directed edges is the deadlock model that holds for the computation [14]. In essence, what a deadlock model does is to specify rules for vertices that are not *sinks* in G to become sinks. (A sink is a vertex with out-degree zero).

We are concerned with *stable properties*. Once a stable property takes hold of a group of processes, only an external intervention that eventually follows its detection can break it. Whenever we refer to G, we mean the wait-for graph that corresponds to a "snapshot" of the distributed computation in the usual sense of a consistent global state [13, 35].

We show that combinatorial concepts from graph convexity have a close relation to stable properties of a wait-for graph in the AND-OR model. In essence, such properties can be regarded as structures associated with graph convexity invariants whose definitions are based on the wait-conditions in the AND-OR model. Some examples are the size of a maximum knot (b-knot [14]), or the minimum number of processes that must become sinks in order to eliminate deadlock. Our studies consider convexity properties of wait-for graphs, directed graphs, and their undirected underlying graphs. The concepts of graph convexity and AND-OR deadlock model will be detailed later.

3.1.1 The AND-OR Model

Let G be a directed graph with vertex set $V(G) = \{v_1, v_2, \ldots, v_n\}$. For $v_i \in V(G)$, let $N^+_G(v_i)$ denote the set of *descendants* of v_i in G (nodes that are reachable from v_i, including itself) and $N^-_G(v_i)$ denote the set of *ancestors* of v_i in G (nodes from which v_i is reachable, including itself). Let $D_i \subseteq N^+_G(v_i)$ be the set of immediate descendants of $v_i \in G$ (descendants that are one edge away from v_i) and $A_i \subseteq N^-_G(v_i)$

its set of immediate ancestors in G (ancestors that are one edge away from v_i). Nodes in $N^+_G(v_i) \setminus N^-_G(v_i)$ are called *subordinates* of v_i.

As in [14], a *deadlock model* for the distributed computation underlying G is a collection of subsets $W_i^1, \ldots, W_i^{p_i}$ of D_i for all $v_i \in V(G)$, where:

- $W_i^1 \cup \cdots \cup W_i^{p_i} = D_i$;
- No two nonempty sets in $W_i^1, \ldots, W_i^{p_i}$ are such that one is a subset of the other;
- In order to exit its blocked state and proceed with its local computation, a node v_i for which $D_i \neq \emptyset$ must receive a signal from all nodes in at least one of the nonempty sets $W_i^1, \ldots, W_i^{p_i}$.

At this level of generality, the deadlock model is known as the *AND-OR model*, reflecting the need for v_i to be signaled by all members of W_i^1 (if nonempty), or all members of W_i^2 (if nonempty), and so on. If at most one of $W_i^1, \ldots, W_i^{p_i}$ is nonempty for all $v_i \in V(G)$, then the deadlock model is the *AND model*. Similarly, if all nonempty sets in $W_i^1, \ldots, W_i^{p_i}$ are singletons for all $v_i \in V(G)$, then the deadlock model is known as the *OR model*. A sufficient condition for the existence of a deadlock in the AND model is the same as the general necessary condition mentioned earlier, that is, the existence of a directed cycle in G. For the OR model, a necessary and sufficient condition is the existence of a *knot* in G. A knot is a subset $K \subseteq V(G)$ with $|K| > 1$ such that $N^+_G(v_i) = K$, for all $v_i \in K$. For details on these conditions and related material, the reader is referred to [94, 131] and the references therein.

Situations that can be characterized by the AND-OR model are, for example, those in which v_i perceives several conjunctions of resources as equivalent to one another, and issues requests for several of them with provisions to withdraw some of them later [14, 27, 96, 127].

We consider in our analysis a simplified AND-OR model in which there are two types of processes, AND and OR. An AND process v_i can only become a sink when its wait is relieved by all the processes in D_i, whereas an OR process v_i can be released by any positive number of processes in D_i. Simplified AND-OR model generalizes AND model and OR model, and although it is a natural subcase of AND-OR model, it is easy to see that a system in the general AND-OR model can be transformed in polynomial-time into a system operating according to the simplified AND-OR model.

For short, we simply say *and/or graph* to refer to a wait-for graph in the simplified AND-OR model. And/or graphs are a well-known data structure in the literature, with many applications in different fields of Computer Science. For instance, artificial intelligence [114, 130], game theory, robotics, operational research, automation and software engineering. And/or graphs model cutting problems [57, 107], robotic task plans [33], assembly/disassembly sequences [54], evaluation of boolean formulas [99], failure dependencies [15], software versioning [45], game trees [97], composition of web services [104] and distributed systems [12]. The optimization problem of minimizing the solution subgraph of an edge-weighted and/or graph was considered in [58, 128], where complexity aspects from both the classical and parameterized point of view are dealt with.

3.1.2 The And/Or-Convexity

As stated earlier, we link the study of and/or graphs with *convexity on graphs*, a discipline that has received broad attention recently, specially for the *geodetic*, *monophonic*, and P_3 convexities [8, 29, 32, 44, 52, 61–63].

We define a family $\mathcal{C}^*$ of vertex subsets of an undirected underlying and/or graph G as follows:

($*$) A set C is a member of $\mathcal{C}^*$ if and only if every or-vertex in $V(G) \setminus C$ does not have neighbors in C and every and-vertex in $V(G) \setminus C$ has at least one neighbor in $V(G) \setminus C$.

Analogously, for the directed version, we define a family $\mathcal{C}^{**}$ of vertex subsets of an and/or graph G as follows:

($**$) A set C is a member of $\mathcal{C}^{**}$ if and only if every or-vertex in $V(G) \setminus C$ does not have *out-neighbors* in C and every and-vertex in $V(G) \setminus C$ has at least one *out-neighbor* in $V(G) \setminus\ C$.

Now, we prove that $(G, \mathcal{C}^*)$ and $(G, \mathcal{C}^{**})$ are graph convexities, called *and/or-convexities* on undirected and directed graphs, respectively. The main goal of this work is to relate the convexity parameters defined below with stable properties of a *distributed computation in the AND-OR model* that uses an and/or graph as the network topology.

Theorem 3.1 $(G, \mathcal{C}^*)$ *defines a graph convexity.*

Proof. By ($*$), it is clear that $\emptyset \in \mathcal{C}^*$ and $V(G) \in \mathcal{C}^*$. To prove that $\mathcal{C}^*$ is closed under intersections, consider two sets in $\mathcal{C}^*$, A and B, and prove that $C = A \cap B$ is in $\mathcal{C}^*$. Let $v \notin A$. If v is an or-vertex, then v has no neighbor in A, and thus in C. If v is an and-vertex, then $N_G(v) \nsubseteq A$, and thus $N_G(v) \nsubseteq C$. By symmetry on the vertices no in B, it follows that $C \in \mathcal{C}^*$. □

Corollary 3.2 $(G, \mathcal{C}^{**})$ *defines a graph convexity.*

Proof. The proof is similar to Theorem 3.1. □

In terms of applications to the convexity parameters presented in Chapter 1, we can cite the following ones:

- A convex set C can be viewed as a set of processes with the following property: relieving all the processes in C from their wait condition, with the corresponding dispatch of grant messages, cannot relieve any process $w \notin C$ from its wait condition. In addition, for every subset of processes S, the minimum convex set C containing S is the set of all processes that can be transitively relieved by those of S. Hence, a hull set S is a minimum set of processes such that all processes in $V(G) \setminus S$ can be transitively relieved by those of S.

- A b-knot is a structure in G that can be used to characterize deadlocks under the AND-OR model [14]. As can be observed in Chapter 1, the convexity number of a directed graph G and the size of a minimum b-knot of G, in fact, are complementary quantities. Hence, computing $cx(G)$ is equivalent to calculating the size of a minimum b-knot of G, which is $n - cx(G)$.

- Any process can be transitively relieved by an appropriated subset of processes of G. The Carathéodory number of G, $c(G)$, means that each process p of G can be transitively relieved by a subset of processes X_p with $|X_p| \leq c(G)$.

- Computing the interval number and the hull number of undirected graphs are special cases of such problems on directed graphs (an undirected edge can be regarded as a pair of directed edges, one for each direction). In practice, the interval number expresses the smallest set of processes that must send grant messages at the initial step of the computation such that all the remaining processes are relieved from their wait condition in exactly one time step. On the other hand, the hull number expresses the minimum number of processes that must send grant messages at the initial step of the computation such that all remaining processes are eventually relieved from their wait condition.

3.2 Computing the Convexity Parameters

In this section, we prove a number of results on the classical convexity parameters defined previously. Remember tat we suppose only connected graphs in this paper. We need some additional definitions. An *and/or graph* G is a graph whose all vertices are labelled as one of {and,or}. Let G_{or} (resp. G_{and}) denote the induced subgraph defined by all the or-vertices (resp. and-vertices) of G. Moreover, let G^i_{or}

be the i-th-connected component of G_{or}, $1 \leq i \leq \omega(G_{\text{or}})$, such that $\omega(G_{\text{or}})$ denotes the number of connected components of G_{or}. We will refer to G always as an and/or graph in the remainder of the section.

3.2.1 The Convexity Number

In this subsection we prove that there exists a linear time algorithm which computes $cx(G)$ for every undirected graph G. Let $\theta(G) = \min_{1\leq i\leq\omega(G_{\text{or}})} |N_G[V(G^i_{\text{or}})]|$ denote the smallest cardinality of the closed neighborhood among all the or-connected components of G.

Theorem 3.3 *Given an undirected connected and/or-graph G, it follows that*

$$cx(G) = \begin{cases} 0 & , \textit{ if } \; V(G_{\text{and}}) = \emptyset; \\ n-2 & , \textit{ if } \; V(G_{\text{and}}) \neq \emptyset \textit{ and } V(G_{\text{and}}) \textit{ is not an independent set}; \\ n-\theta(G) & , \textit{ otherwise.} \end{cases}$$

Furthermore $cx(G)$ can be computed in linear time.

Proof. If G does not contain any and-vertex, then $I_t[\{v\}] = V(G)$ for every vertex v and some time step t. Thus the convex sets of G are only the trivial convex sets and thus $cx(G) = 0$. Hence, suppose that there must exist at least one and-vertex in $V(G)$. For every vertex v, the subset $V(G) \setminus \{v\}$ is not convex, implying that $cx(G) \leq n-2$. If there exists an edge (u, v) such that u and v are and-vertices, then $V(G) \setminus \{u, v\}$ is a convex set, since u and v are neighbors outside $V(G) \setminus \{u, v\}$. Thus, we get that $cx(G) = n - 2$.

Let us suppose now that $V(G_{\text{and}})$ induces an independent set. Note that a non-trivial convex set cannot contain all the or-vertices of G. Moreover, if an or-vertex v does not belong to a non-trivial convex set C then no vertex of the connected component G^i_{or} containing v could be in C either. Furthermore, if an and-vertex $v \in N_G(V(G^i_{\text{or}}))$ belongs to C, then C must include the entire or-component G^i_{or}. Hence $N_G[V(G^i_{\text{or}})]$ must be in $V(G) \setminus C$ and the maximum cardinality of C is equal to $n - \theta(G)$. It is not hard to see that computing $\theta(G)$ can be done in $\Theta(n + m)$ time. □

Corollary 3.4 *The convexity number of directed connected graphs can be computed in polynomial time.*

Proof. Let G be a directed graph. If G has a sink vertex v, then $V(G) \setminus \{v\}$ is a maximum non-trivial convex set. Otherwise, if G has only and-vertices, then $V(G) \setminus C_k$ is a maximum non-trivial convex set, where C_k is a minimum directed cycle of G. Such a cycle can be found in polynomial time by adapting the Floyd-Warshall algorithm [46]. If G has both and- and or- vertices, then $V(G) \setminus B$ is a maximum non-trivial convex set, where B is a minimum b-knot of G, which can also be found in polynomial time [14]. □

3.2.2 The Interval Number

Computing the interval number of an and/or graph G is NP-hard in general. If all vertices are labelled "or" then determining $in(G)$ is equivalent to finding the cardinality of a minimum *dominating set* of G, denoted by $\gamma(G)$, which is an NP-hard problem even for split or bipartite graphs [20]. On the other hand, if all vertices are labelled "and" then computing $in(G)$ is equivalent to determining the cardinality of a minimum vertex cover of G. This is true because, for each pair of adjacent vertices u and v that are not in a subset $S \subset V(G)$, they cannot be in $I_t[S]$, for any $t > 0$. Thus, $V(G) \setminus S$ must be an independent set in G, for any interval set S, which implies that S is a vertex cover of G. Since it is NP-hard to compute $\beta(G)$ even for cubic planar graphs [75], computing $in(G)$ is also NP-hard.

Therefore, every interval set S must contain a vertex cover of G_{and} and every vertex in $V(G) \setminus S$ has a neighbor in S. Moreover, every minimum vertex cover of G is an interval set of G. From these observations it follows that

$$\min\{\gamma(G), \beta(G_{\text{and}})\} \ \leq \ in(G) \ \leq \ \beta(G).$$

Next, we consider the complexity of determining $in(G)$ for general bipartite graphs. We consider the following decision problem:

Interval Number
Input: An and/or graph G and an integer $k \geqslant 1$.
Question: Is $in(G) = k$?

Theorem 3.5 Interval Number *is NP-complete for connected bipartite graphs with exactly one* and*-vertex.*

Proof. We consider a reduction from the Vertex Cover problem for cubic graphs with at least six vertices, which remains NP-complete [75]. Given a cubic graph $G = (V, E)$, we construct a bipartite graph $G' = (V', E')$ such that

$$V(G') = \{w_{v_i} : v_i \in V(G)\} \cup \{w_{v_iv_j} : v_iv_j \in E(G)\} \cup \{w\}$$

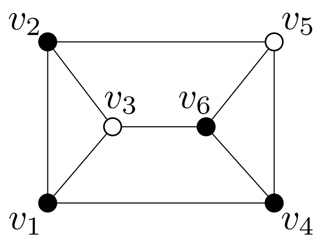
and

$$E(G') = \{ww_{v_i} : w_{v_i} \in V(G')\} \cup \{w_{v_i}w_{v_lv_r} : i = l \text{ or } i = r, \text{ and } w_{v_i}, w_{v_lv_r} \in V(G')\}.$$

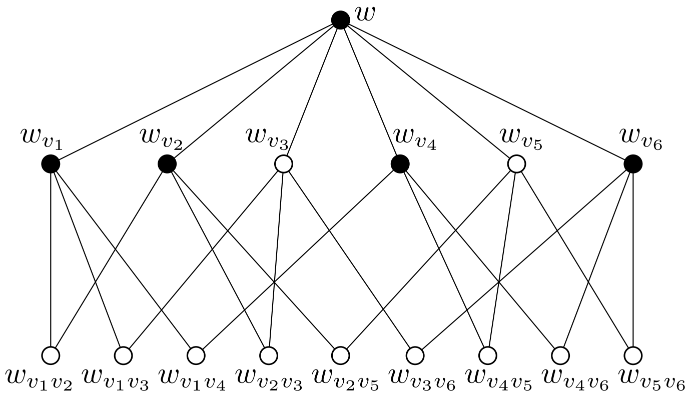
Finally, assign "and" to w and "or" to the remaining vertices of $V'(G')$. Figure 3.1 shows an example of the construction. Observe that G' is a bipartite graph with $1+5n/2$ vertices and $4n$ edges. Its parts are defined by $\{w\} \cup \bigcup_{v_iv_j \in E(G)} \{w_{v_iv_j}\}$ and $\bigcup_{v_i \in V} \{w_{v_i}\}$.

Now we will prove that $in(G') = \beta(G)+1$. Notice that the set $S = \{w\} \cup C$ is an interval set of G', where C is a minimum vertex cover of G, since every vertex $w_{v_iv_j}$ has a neighbor $w_{v_i} \in C$ and every vertex w_{v_i} is adjacent to w. Thus $in(G') \leq |S| = \beta(G)+1$.

Let S be a minimum interval set of G'. If $w \notin S$ then every vertex w_{v_i} must be in S, implying that $in(G) = n$, since S is an interval set of G'. Therefore $n \leq \beta(G)+1$, which implies that G is a clique. In other words, $G = K_4$, a contradiction by the assumption $n \geq 6$. Hence $w \in S$ and we can obtain a new interval set S' from S as follows. For every $w_{v_\ell v_r} \in S$, if both w_{v_ℓ} and w_{v_r} are not in S, replace $w_{v_\ell v_r}$ by either w_{v_ℓ} or w_{v_r}. Otherwise, just remove $w_{v_\ell v_r}$ of S. We can see that S' is an interval set of G' smaller than S. Moreover S' is also a minimum vertex cover of G. Hence $\beta(G)+1 = |S'| \leq |S| = in(G')$, completing the proof. □



(a) A cubic graph G.



(b) Bipartite graph G' obtained from G.

Figure 3.1: Example of transformation from graph G to G' in Theorem 3.5. The black vertices are a minimum vertex cover of G in Figure 3.1a and a minimum interval set of G' in Figure 3.1b.

Corollary 3.6 *Determining the interval number of a directed connected graph G is NP-hard.*

Proof. Given an undirected and/or graph G, we can construct a directed and/or graph G' replacing each edge $e = (u, v)$ by two arcs $a_e^1 = (u, v)$ and $a_e^2 = (v, u)$. Assuming that G has only and-vertices, it is easy to see that G has a vertex cover of size k if and only if G' has an interval set of size k. Note that, if G has only or-vertices we obtain a reduction from the dominating set problem. □

Approximation Algorithm

Algorithm 2 describes an approximation procedure for the interval number. Given a connected and/or graph G, a set S is an interval set if and only if

- $G \setminus S$ does not contain any edge uv, where u or v is an and-vertex, and
- every or-vertex in $V(G) \setminus S$ has a neighbor in S.

The first condition implies that if G_1 is the spanning subgraph of G that contains all edges uv of G where u or v is an and-vertex, then $in(G) \geq \beta(G_1)$. The second condition implies $in(G) \geq \gamma(G, V(G_{\text{OR}}))$, where $\gamma(G, V(G_{\text{OR}}))$ is the minimum cardinality of a set D of vertices of G such that every vertex in $V(G_{\text{OR}}) \setminus D$ has a neighbor in D. The union of a 2-approximate vertex cover of G_1 and of a $\log|V(G_{\text{OR}})|$-approximate $V(G_{\text{OR}})$-dominating set of G yields a $(2 + \log|V(G_{\text{OR}})|)$-approximate interval set of G.

In fact, one can easily construct an equivalent set cover instance in order to carry over to $\gamma(G, V(G_{\text{OR}}))$, where the ground set to be covered is the set of or-vertices, and the sets are the closed neighborhoods of all vertices of G.

Algorithm 2: A $(2 + \log|V(G_{\text{OR}})|)$-approximation algorithm for $in(G)$.

```
Data: A connected and/or Graph $G$.
Result: An interval set $S$ of $G$ of size at most $(2 + \log|G_{\text{OR}}|) * in(G)$.
1 $G_1 \leftarrow$ A spanning subgraph of $G$ that contains all edges $uv$ of $G$ where $u$ or $v$ is an
   and-vertex;
2 $S \leftarrow$ The matched vertices of a maximal matching in $G_1$;
3 $U \leftarrow V(G_{\text{OR}})$;
4 while $U \neq \emptyset$ do
5     $v \leftarrow$ The vertex of $G$ whose $N_G[v]$ contains the largest number of uncovered or-vertices;
6     Cover all vertices in $N[v] \cap V(G_{\text{OR}})$;
7     $U \leftarrow U \setminus N[v]$;
8     $S \leftarrow S \cup \{v\}$;
Result: $S$
```

Theorem 3.7 *Given a connected and/or graph G, Algorithm 2 is a $(2 + \log|V(G_{\text{OR}})|)$-approximation algorithm for $in(G)$. Furthermore computing $in(G)$ is a Log-APX-complete problem.*

Proof. Let us denote the size of an optimum solution by OPT and the size returned by Algorithm 2 for an instance I by $A(I)$. We know that every interval set of G must include a vertex cover of G_{AND}. Thus, line 1 represents the classical 2-approximation algorithm to obtain a vertex cover [135]. After the line 2 it is obtained a set S_1 that covers all edges between and-vertices. Since any interval set must be a vertex cover for G_{and}, it follows that

$$|S_1|/2 \leq OPT \Longrightarrow |S_1| \leq 2 * OPT. \tag{3.1}$$

Lines 3–8 present the $\log n$-approximation algorithm for Dominating Set [135], where U represents the universal set and the sets are composed by $|N_G[v]|$, for every $v \in V(G)$. Hence, all or-vertices are covered by a set S_2, following that

$$|S_2| \leq \log(|V(G_{\text{OR}})|) * OPT. \tag{3.2}$$

From Eq. (3.1) and Eq. (3.2) we obtain the following for any instance I:

$$\begin{aligned} A(I) &= |S_1| + |S_2| \\ &\leq 2 * OPT + \log(|V(G_{\text{OR}})|) * OPT \\ &= (2 + \log|V(G_{\text{OR}})|) * OPT. \end{aligned}$$

Since Dominating Set is in Log-APX-complete [68], the theorem follows. □

Corollary 3.8 *Algorithm 2 is a $(1 + \log|V(G_{\text{OR}})|)$-approximation for $in(G)$, if G is a bipartite connected and/or graph. Moreover, if G is such that one part has only* and*-vertices and the other has only* or*-vertices, then $in(G)$ can be computed in polynomial time.*

Proof. By the well-known König-Egerváry's theorem, a minimum vertex cover for G can be found in polynomial time for bipartite graphs. Thus the first step in Algorithm 2 results in a vertex cover for the and-vertices of size at most OPT. Hence the approximation is limited by the dominating step of the or-vertices, obtaining an approximation of $(1 + \log|V(G_{\text{OR}})|)$.

When G has each part formed by vertices of the same type, distinct from the type of the other part, we can see that every interval set S must be a minimum vertex cover of G. This is true because no edges can exist between vertices of $V(G) \setminus S$. Hence the proof follows by the König-Egerváry's theorem. □

A Linear Time Algorithm for Trees

Now we prove that Interval Number is polynomially solvable for trees. We adapt the classical linear time algorithm presented in [42], which computes $\gamma(T)$ for a tree T. In fact, the algorithm works for a more general domination called *mixed domination*, defined as follows. Consider a partition P of the vertices of G into three subsets V_1, V_2, V_3 containing *free*, *bound*, and *required* vertices, respectively. Such a partition P is called an *m-partition*. A *mixed dominating set* D of a graph G *with respect to* P (or simply *wrt* P) is a subset of vertices containing all required vertices and such that every bound vertex either belongs to D or is adjacent to some vertex of D. Furthermore, we require an additional restriction on each and-vertex v: $v \notin D$

if and only if $N_G(v) \subseteq D$. We denote the size of a minimum mixed dominating set of G wrt P by $\gamma_m(G, P)$.

Proposition 3.9 *There exists an m-partition P of a connected graph G such that $in(G) = \gamma_m(G, P)$.*

Proof. Let S be a minimum interval set of G. This implies that every or-vertex of $V(G) \setminus S$ has at least one neighbor in S and all and-vertices in $V(G) \setminus S$ have their entire neighborhood in S. By setting $V_1 = V_3 = \emptyset$ and $V_2 = V(G)$ we obtain an m-partition P such that S is a mixed dominating set D of G wrt P. This shows that $\gamma_m(G, P) \leq in(G)$. Now, let D be a minimum mixed dominating set of G wrt P. Clearly, every and-vertex v of $V(G) \setminus D$ has all of its neighbors in D, and every or-vertex of $V(G) \setminus D$ is covered by at least one vertex of D. Hence D is also an interval set of G, which implies $in(G) \leq \gamma_m(G, P)$. □

The next theorem is the basis of the algorithm that computes $\gamma_m(T, P)$ for a tree T. We assume without loss of generality that all leaves of T are or-vertices. Denote by $G - v$ the graph obtained by removing v from G. Moreover, for each and-vertex v, let $r(v)$ denote the number of its required neighbors.

Theorem 3.10 *Let T be a tree with an m-partition P, v a leaf of T, u the neighbor of v, and $T' = T - v$. Let P_v be the m-partition of T' obtained by removing v from the member of P that contains v. Then:*

1. *if $v \in V_1$ and $u \in V(T_{\text{and}})$ then $\gamma_m(T, P) = \gamma_m(T', P')$, where P' is the m-partition obtained from P_v by setting u as required;*
2. *if $v \in V_2$ then $\gamma_m(T, P) = \gamma_m(T', P')$, where P' is the m-partition obtained from P_v by setting u as required;*
3. *if $v \in V_3$ and $u \in V_3$ then $\gamma_m(T, P) = 1 + \gamma_m(T', P_v)$;*
4. *if $v \in V_3$ and $u \notin V_3$ then $\gamma_m(T, P) = 1 + \gamma_m(T', P'')$, where P'' is the m-partition of T' obtained by setting u as free if either $u \in V(T_{\text{or}})$ or $u \in V(T_{\text{and}})$ and $r(u) = |N_T(u) \cap V(T)|$.*

Proof. (1) Since $v \in V_1$, any mixed dominating set of T' wrt P' is also a mixed dominating set of T wrt P, and therefore $\gamma_m(T, P) \leq \gamma_m(T', P')$. To prove that $\gamma_m(T', P') \leq \gamma_m(T, P)$, let D be a minimum mixed dominating set of T wrt P. If $v \notin D$ then D is also a mixed dominating set of T' wrt P'. Thus, suppose that $v \in D$; this implies $u \notin D$ (otherwise $D \setminus \{v\}$ is a mixed dominating set of T wrt P, contradicting the optimality of D). Let $D' = (D \setminus \{v\}) \cup \{u\}$. If $u \in V(T_{\text{and}})$

then D' is a mixed dominating set of T' wrt P', because $v \notin (N_T(u) \cap D')$. Hence $\gamma_m(T', P') \leq |D'| = |D| = \gamma_m(T, P)$.

(2) Since u is required in T', every mixed dominating set of T' wrt P' is also a mixed dominating set of T wrt P, implying that $\gamma_m(T, P) \leq \gamma_m(T', P')$. The proof of inequality $\gamma_m(T', P') \leq \gamma_m(T, P)$ follows as in item (1).

(3) The proof of this case is trivial.

(4) Let D' be a mixed dominating set of T' wrt P''. From the definition of P'', it is clear that $D \cup \{u\}$ is a mixed dominating set of T wrt P. Therefore $\gamma_m(T, P) \leq 1 + \gamma_m(T', P'')$. Now, let D be a minimum mixed dominating set of T wrt P. Since v is required, $v \in D$. Moreover, if $u \in D$ then $D \setminus \{v\}$ is a mixed dominating set of T' wrt P''. Finally, assume $u \notin D$. Since u is free in T', we have two cases: (a) if u is an or-vertex then u is dominated for some vertex $w \in D \setminus \{u, v\}$; (b) if u is an and-vertex then $r(u) = |N_T(u) \cap V(T)|$. In both cases it is clear that $D \setminus \{v\}$ is a mixed dominating set of T' wrt P''. Hence, $\gamma_m(T', P'') \leq |D| - 1 = \gamma_m(T, P) - 1$.
□

Theorem 3.10 directly leads to the following linear time procedure to compute a minimum mixed domination set for a labelled tree:

Algorithm 3: Linear time algorithm that computes $\gamma_m(T, P)$ for a tree T.

```
Data: An and/or tree $T$ whose vertices are labelled as free, bound, or required.
Result: A minimum mixed dominating set MDS of $T$.
1  MDS $\leftarrow \emptyset$;
2  for every $v \in V(T_{\text{and}})$ do
3      $r(v) \leftarrow 0$;
4  repeat
5      $v \leftarrow$ a leaf of $T$;
6      $u \leftarrow$ neighbor of $v$ in $T$;
7      if (($v$ is free and $u \in V(T_{\text{and}})$) or $v$ is bound ) and $u$ is not required then
8          set $u$ as required;
9          for every $w \in N_T(u) \cap V(T_{\text{and}})$ do
10             $r(w) \leftarrow r(w) + 1$;
11     else
12         if $v$ is required then
13             MDS $\leftarrow$ MDS $\cup \{v\}$;
14             if $u$ is not required and
                 ($u \in V(T_{\text{or}})$ or ($u \in V(T_{\text{and}})$ and $r(u) = |N_T(u) \cap V(T)|$)) then
15                 set $u$ as free;
16     $T \leftarrow T - v$;
17 until $n = 1$;
18 if the last vertex $v$ is not free then
19     MDS $\leftarrow$ MDS $\cup \{v\}$;
```

In the above algorithm, $r(v)$ is initially set to zero for each and-vertex v. Lines 7–10 cover cases (1) and (2) in Theorem 3.10, whereas lines 11–15 cover cases (3) and (4). If $v \in V_1$ then v may or may not belong to a mixed dominating set of T.

Thus, if v is an or-vertex then it is either a leaf of T or has a required neighbor already processed. On the other hand, if v is an and-vertex then $r(v) = d(v)$, and therefore v is the last vertex in the computation. If $r(v) = d(v) - 1$ then v is a bound vertex, since v depends on whether $u \in N_T(v)$ is in D or not, where u has not been analyzed yet.

3.2.3 The Hull Number

In this subsection we present some complexity and structural properties on hull sets of and/or graphs G. Computing $hn(G)$ is trivially solvable if G contains only or-vertices – in this case every vertex defines itself a hull set. However, as described in Subsection. 3.2.2, if G contains no or-vertex then determining $hn(G)$ is an NP-hard problem, since $\beta\left(G_{\text{and}}\right) \leq hn(G) \leq in(G)$ and so $hn(G) = \beta(G) = \beta\left(G_{\text{and}}\right)$. Based on this fact, we obtain the following result:

Lemma 3.11 *Let G be a connected graph containing* and-*vertices. For any hull set S of G, there exists a hull set S' with no* or-*vertices and such that $|S'| \leq |S|$.*

Proof. Let S be a hull set of G and let $v \in S$ be an or-vertex. If $N_G(v) \cap S \setminus \{v\} \neq \emptyset$, then $S' = S \setminus \{v\}$ is also a hull set of G. Otherwise, if there exists at least one and-vertex $u \in N_G(v) \cap (V(G) \setminus S)$, then $S' = \{u\} \cup S \setminus \{v\}$ is also a hull set of G, since $S \subset I_1[S']$. Finally, assume that $N_G(v)$ is entirely contained in G^i_{or}. Thus, if $N_G(V(G^i_{\text{or}}) \setminus \{v\}) \cap S \neq \emptyset$, then $S' = S \setminus \{v\}$ is a hull set of G. Otherwise, all the and-neighbors of G^i_{or} have at least one neighbor outside S. Hence let $S' = \{u\} \cup S \setminus \{v\}$, where u is such an and-neighbor of $N_G(V(G^i_{\text{or}}) \setminus \{v\})$. It is easy to see that S' is a hull set of G. Applying the same reasoning to each or-vertex $v \in S$ we obtain a new hull set as required in the statement. □

Let G^* be the and/or graph obtained from G contracting each or-component G^i_{or} to a unique vertex v^i, for all $1 \leq i \leq \omega\left(G_{\text{or}}\right)$. In other words, replace $V\left(G^i_{\text{or}}\right)$ by v^i such that $N\left(v^i\right) = N\left(V(G^i_{\text{or}})\right) \cap V\left(G_{\text{and}}\right)$. We say that G^* is the *contracted graph* of G. Moreover, for each subset $S \subseteq V(G)$ let us define the *contracted set* $S^* \subseteq V(G^*)$ of S as follows:

- $v \in S \cap V(G_{\text{and}})$ if and only if $v \in S^* \cap V(G_{\text{and}})$;
- if $v \in S \cap V(G^i_{\text{or}})$ then $v^i \in S^*$;
- if $v^i \in S^*$ then there exists at least one vertex $u \in S \cap V(G^i_{\text{or}})$.

Observe that the or-vertices of G^* induce an independent set.

Lemma 3.12 *Let G^* be the contracted graph of G. For every $S \subseteq V(G)$, $H(S) \cap V(G_{\text{and}}) = H(S^*) \cap V(G_{\text{and}})$.*

Proof. If S does not contain any or-vertex, then $S^* = S$ and the lemma is trivial. Since every and-vertex of S is in S^*, and vice versa, let $v \in H(S) \cap (V(G_{\text{and}}) \setminus S)$. Thus $N_G(v) \subseteq I_t[S]$ for some $t \in \mathbb{N}$, and suppose by contradiction that $v \notin H(S^*)$. For every $u \in N_G(v) \cap V(G^i_{\text{or}})$, we get that $u \in I_{t^*}[S]$ for some $t^* \leq t$, implying that $S \cap N[V(G^i_{\text{or}})] \neq \emptyset$. Hence $S^* \cap N[v^i] \neq \emptyset$ and thus $v^i \in H[S^*]$. Therefore, there must exist an and-vertex $w \in N_G(v)$ such that $w \in H(S)$ and $w \notin H(S^*)$. If $w \notin S$ then w and v depend on each other to be in $H(S)$, which implies that $w \notin H(S)$. This means that $w \in S$ and thus $w \in S^*$. Hence $w \in H(S^*)$, a contradiction.

Conversely, consider $v \in H(S^*) \cap (V(G_{\text{and}}) \setminus S^*)$. As observed before, each and-neighbor of v must be in S^* and thus in S. Moreover, each $v^i \in N_G(v)$ belongs to $I_t[S^*]$ for some $t \in \mathbb{N}$. Thus there must exist an or-vertex $u \in V(G^i_{\text{or}})$ such that $u \in H(S)$. Hence $V(G^i_{\text{or}}) \subset H[S]$, i.e., $N_G(v) \subseteq H(S)$ in G. □

Theorem 3.13 *For every connected and/or graph G, $hn(G) = hn(G^*)$.*

Proof. Let S be a minimum hull set of G that contains only and-vertices, which exists by Lemma 3.11. Since all the or-vertices of G must be in $H(S)$, it follows that every or-component of G must be "reached" by S. Thus all the or-vertices of G^* are also reached by S. By Lemma 3.12, S is also a hull set of G^*, implying that $hn(G^*) \leq hn(G)$.

Conversely, let S^* be a minimum hull set of G^* containing only and-vertices. Since all the or-vertices of G^* belong to $H(S^*)$, at least one vertex of each or-connected component of G^* either is in S or is adjacent to one vertex of S. Hence all the or-vertices of G belong to $H(S^*)$. Therefore, by Lemma 3.12, S^* is also a hull set of G, that is, $hn(G) \leq hn(G^*)$. □

By Theorem 3.13, we assume that G is contracted in the remaining of this subsection.

Corollary 3.14 *If $V(G_{\text{and}})$ is an independent set, then computing $hn(G)$ is NP-hard.*

Proof. In this case G is a bipartite graph whose parts are formed by the and-vertices and the or-vertices, respectively. Since every or-vertex must be in $H(S)$ for every hull set S, we get that all and-vertices must also be in $H(S)$. Thus, it follows by Lemma 3.11, Lemma 3.12 and, Theorem 3.13 that $hn(G)$ is the size of a minimum set S of and-vertices which covers all the or-vertices. In other words, computing a minimum hull number is equivalent to determining a minimum Set cover of G, where the or-vertices are the elements and each and-vertex v represents the subset of elements that v is adjacent. Since computing a minimum Set Cover is NP-hard, the corollary follows. □

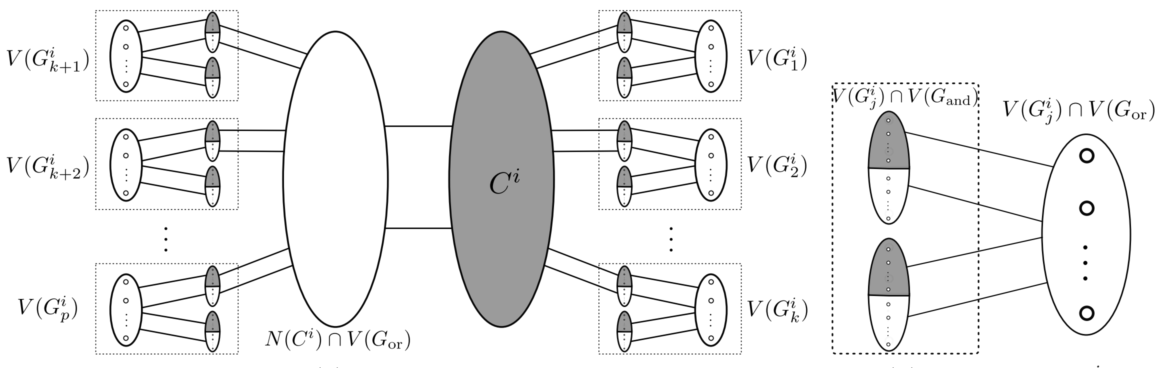
Now, let us consider that the and-vertices does not induce an independent set. Let C^i be the i-th vertex cover of G_{and}. Moreover, let G^i be the subgraph induced by the subset $V(G) \setminus (C^i \cup (N(C^i) \cap V(G_{\text{or}})))$, and let G^i_j denote the j-th connected component of G^i. We use the notation $H' \subseteq H$ to mean that H' is a connected component of graph H. Figure 3.2a shows a representation of a hull set of G by the gray sets.

Theorem 3.15 *The hull number of a connected graph G can be determined as*

$$hn(G) = \min_{C^i} \left\{ |C^i| + \sum_{G^i_j \subseteq G^i} hn\left(G^i_j\right) \right\}.$$

Proof. Let C^i be a vertex cover of G_{and}, and let S denote the set formed by C^i and the union of the minimum hull sets of each G^i_j, that contain only and-vertices by Lemma 3.11. Let us prove that $hn(G) \leq \min\left\{ |C^i| + \sum_{G^i_j \subseteq G^i} hn\left(G^i_j\right) \right\}$. Note that every or-vertex has at least one neighbor in S, otherwise their and-neighbors cannot be in $H(S)$, a contradiction. Furthermore, since $V(G_{\text{and}}) \setminus S$ is an independent set, we get that all the and-vertices are reached in at most two time steps. Hence S is a hull set of G.

Conversely, let S be a minimum hull set of G which contains only and-vertices. We know that S must contain a vertex cover C^i of G_{and}, otherwise the endpoints of an edge between and-vertices are neighbors outside S. Therefore, every connected G^i_j component of G^i is a bipartite graph with parts containing only and-vertices and only or-vertices, respectively. Hence, a minimum hull set of G containing only and-vertices can be defined by taking a vertex cover of G_{and} that minimizes the sum of the hull numbers of the generated connected components. Therefore $hn(G) \geq \min\left\{ |C^i| + \sum_{G^i_j \subseteq G^i} hn\left(G^i_j\right) \right\}$. $\square$


$V(G^i_{k+1})$
$V(G^i_{k+2})$
$V(G^i_p)$
$N(C^i) \cap V(G_{\text{or}})$
C^i
$V(G^i_1)$
$V(G^i_2)$
$V(G^i_k)$
$V(G^i_j) \cap V(G_{\text{and}})$
$V(G^i_j) \cap V(G_{\text{or}})$

(a) Hull set of G.   (b) Component G^i_j.

Figure 3.2: Representation of a hull set of G given by the gray sets.

We can see a minimum hull set of G as a smallest set S of and-vertices that covers all the or-vertices and such that $V(G_{\text{and}}) \setminus S$ is an independent set. Furthermore, a hull set S containing only and-vertices is also an interval set of G if and only if S covers all the or-vertices and every and-vertex outside S has their whole neighborhood in S.

Corollary 3.16 *Algorithm 2 is a $(2 + \log |V(G_{\text{OR}})|)$-approximation algorithm for $hn(G)$. Furthermore computing $hn(G)$ is a Log-APX-complete problem.*

Proof. The proof is analogous to that of Theorem 3.7. □

Corollary 3.17 *Determining the hull number of a directed connected graph G is NP-hard.*

Proof. Assuming that G has only and-vertices the interval and hull numbers are equal. □

Linear Time Algorithm for Trees

Given a tree T, we can adapt Algorithm 3 to find its hull number, where $V(T)$ is partitioned into free, bound and required vertices. However, the definition of mixed domination is slightly different in this case. Formally, a mixed dominating set D^* of T is as follows: D^* contains all required vertices; every bound vertex belongs to D^* or is covered by it; $v \notin D^*$ if and only if $N_G(v) \cap V(G_{\text{and}}) \subseteq D^*$, for every and-vertex v. We denote by $\gamma_m^*(T, P)$ the size of a minimum mixed dominating set of T wrt an m-partition P into free, bound, and required vertices. The following proposition is the analogous of Proposition 3.9 for $\gamma_m^*(G, P)$.

Proposition 3.18 *Let G be a connected and/or graph. Then there exists an m-partition P of G such that $hn(G) = \gamma_m^*(G, P)$.*

Proof. Let S be a minimum hull set of G containing no or-vertices. Then S must cover all the or-vertices of G, and every and-vertex of $V(G) \setminus S$ must have its entire neighborhood in S. Thus S is a mixed dominating set of G wrt the m-partition $P = (\emptyset, V(G), \emptyset)$, and this implies $\gamma_m^*(G, P) \leq hn(G)$. Now, let D^* be a minimum mixed dominating set of G wrt P. Thus $N_G(v) \cap V(G_{\text{and}}) \subseteq D^*$ if and only if $v \notin D^*$, for each and-vertex v. Let w be an or-vertex such that $w \in D^*$. If w is a bound vertex and $N_G(w) \cap D^* \neq \emptyset$ then $D^* \setminus \{w\}$ is also a mixed dominating set of G wrt P, a contradiction. Hence there must exist $u \in N_G(w) \cap (V(G) \setminus D^*)$, and then $(D^* \setminus \{w\}) \cup \{u\}$ is also a minimum mixed dominating set of G wrt P. However, if w is a required vertex, we can set w as bound and apply the same arguments, for each such or-vertex. The set D' obtained in this way contains only and-vertices and every or-vertex is adjacent to some vertex of D'. Hence D' is a hull set of G, implying that $hn(G) \leq |D'| = |D^*| = \gamma^*_m(G, P)$. □

Let us denote by $r^*(v)$ the number of required and-neighbors of v, for every and-vertex v. The following theorem describes how to compute a minimum hull set on trees.

Theorem 3.19 *Let T be a tree with an m-partition P, v a leaf of T, u the neighbor of v, and $T' = T - v$. Let P_v be the m-partition of T' obtained by removing v from the member of P that contains v. Then:*

1. *if $v \in V_1$ then $\gamma^*_m(T, P) = \gamma^*_m(T', P_v)$;*
2. *if $v \in V_2$ then $\gamma^*_m(T, P) = \gamma^*_m(T', P')$, where P' is the m-partition obtained from P_v by setting u as required;*
3. *if $v \in V_3$ and $u \in V_3$ then $\gamma^*_m(T, P) = 1 + \gamma^*_m(T', P_v)$;*
4. *if $v \in V_3$ and $u \notin V_3$ then $\gamma^*_m(T, P) = 1 + \gamma^*_m(T', P'')$, where P'' is the m-partition of T' obtained by setting u as free if either $u \in V(T_{\text{or}})$ or $u \in V(T_{\text{and}})$ and $r^*(u) = |N_T(u) \cap V(T_{\text{and}})|$.*

The proof of Theorem 3.19 is very similar to that of Theorem 3.10, and its proof can be found in Appendix 5.1.5, as well as a procedure that computes in linear time the hull number for trees.

3.2.4 The Carathéodory Number

Now, we present results on the Carathéodory number in the and/or-convexity.

Lemma 3.20 *Given a connected and/or graph G, it follows that*

$$c(G) = \begin{cases} 1 & \text{, if } G \text{ contains only or-vertices;} \\ \Delta(G) & \text{, if } G \text{ contains only and-vertices.} \end{cases} \tag{3.3}$$

Proof. Since each vertex defines a hull set of G, if G contains only or-vertices, the theorem trivially holds in this case. Assume that G contains only and-vertices. The convex hull of any subset $S \subset V(G)$ must contain the entire neighborhood of each vertex in $H(S) \setminus S$. This implies that $c(G) \leq \Delta(G)$. Let $S = N_G(v)$, where $d(v) = \Delta(G)$. The convex hull of every subset $X \subset S$ cannot contain v, although $v \in H(S)$. Hence $c(G) \geq \Delta(G)$. □

Lemma 3.20 implies that $c(G) \geq \max\{1, \Delta(G_{\text{and}})\}$, where $\Delta(G_{\text{and}})$ denotes the maximum degree of G_{and}. Next we present a polynomial-time algorithm that computes $c(G)$ for every graph G. Let $d_{\text{and}}(v)$ denote the number of and-neighbors of v. In addition, given an and-vertex v, let $\overline{d_{\text{or}}}(v)$ denote the number of or-neighbors of v that are not adjacent to any and-vertex in the neighborhood of v. Formally, $\overline{d_{\text{or}}}(v) = \left| N_{\text{or}}(v) \setminus \bigcup_{u \in N_{\text{and}}(v)} N_G(u) \right|$.

Theorem 3.21 *Given a connected and/or graph G, it follows that*

$$c(G) = \max\{1, \max_{v \in V(G_{\text{and}})} \{d_{\text{and}}(v) + \overline{d_{\text{or}}}(v)\}\}.$$

Furthermore, computing $c(G)$ can be done in linear time.

Proof. By Lemma 3.20, if G contains only or-vertices then $c(G) = 1$ and the theorem trivially holds. Hence, let $S \subseteq V(G)$ containing both and-vertices and or-vertices. Since the or-vertices in $H(S)$ can be obtained by taking all subsets $X \subseteq S$ with $|X| = 1$, we can consider only the and-vertices in $H(S) \setminus S$. Thus, let v be such an and-vertex. As in the proof of Lemma 3.20, every and-neighbor of v must be in any subset $X \subset S$ such that $v \in H(X)$. Also, the entire neighborhood of v should satisfy this condition. Thus, if there exists an or-neighbor w of v which is not adjacent to any and-neighbor of v, w must be included in X. Hence $|X| \geq d_{\text{and}}(v) + \overline{d_{\text{or}}}(v)$ and it is clear to see that $c(G) \geq \max_{v \in V(G_{\text{and}})}\{d_{\text{and}}(v) + \overline{d_{\text{or}}}(v)\}$. Furthermore, it is not hard to see that $c(G) \leq \max_{v \in V(G_{\text{and}})}\{d_{\text{and}}(v) + \overline{d_{\text{or}}}(v)\}$, otherwise there must exist an and-vertex that is not in $H(S)$, which contradicts the maximality of $d_{\text{and}}(v) + \overline{d_{\text{or}}}(v)$. Finally, it is not hard to see that computing $c(G)$ can be done in $\Theta(n + m)$ time, which concludes the proof. □

3.3 Generalized Threshold Processes on Graphs

"Everything's better with some wine in the belly."
(Tyrion Lannister)

— G. Martin, *A Game of Thrones*

Many irreversible processes typically consider a set of vertices of a given graph that grows iteratively according to certain extension rules. These rules usually involve threshold values, which may be vertex-dependent. A specific vertex might for instance enter some set of 'infected' vertices provided that sufficiently many of its neighbors are already contained in that set. In the present section we consider a generalization of such processes, where we replace the threshold values by specific sets, that is, each vertex may individually specify certain subsets of its neighborhood, and the process is guided by these specific selections.

For an integer k and a set V, let $[k]$ be the set of all positive integers at most k.

Based on discrete convexity notions [30, 60], now we give formal definitions for the generalized threshold processes.

Let G be a graph. A *threshold function* on G is a function $\tau : V(G) \to 2^{V(G)}$ such that $\tau(u) \subseteq 2^{N_G(u)}$ for every vertex u of G.

Let S be a set of vertices of G. The *τ-interval* $I_\tau(S)$ of S is the set

$$S \cup \{u : u \in V(G) \setminus S \text{ and } N_G(u) \cap S \in \tau(u)\},$$

that is, $I_\tau(S)$ contains S as well as all vertices u from the complement of S whose neighborhood intersects S in a set that is contained in $\tau(u)$. For example, in Figure 3.3 let τ such that $\tau(v_1) = \{\emptyset, \{v_2\}\}$, $\tau(v_2) = \{\{v_1\}, \{v_1, v_3\}\}$, and $\tau(v_3) = \{\emptyset\}$. Considering $S = \{v_2\}$, we can see that $I_\tau(S) = \{v_1, v_2\}$, since $\{v_2\} \in \tau(v_1)$, and $\{v_2\} \notin \tau(v_3)$.

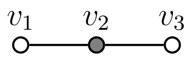


Figure 3.3: Example of a graph and τ function.

Let $I^0_\tau(S) = S$, and, for every positive integer i, let $I^i_\tau(S) = I_\tau(I^{i-1}_\tau(S))$. If k is the smallest non-negative integer with $I^{k+1}_\tau(S) = I^k_\tau(S)$, then $I^k_\tau(S)$ is the *τ-hull* $H_\tau(S)$ of S, and the sequence $\left(I^0_\tau(S), \ldots, I^k_\tau(S)\right)$ is the *τ-threshold process on G starting with S*.

If $I_\tau(S) = S$, then S is *τ-convex*. If $I_\tau(S) = V(G)$, then S is a *τ-interval set*. Finally, if $H_\tau(S) = V(G)$, then S is a *τ-hull set*. The minimum order of a τ-interval set is the *τ-interval number* $i_\tau(G)$ of G, and the minimum order of a τ-hull set is the *τ-hull number* $h_\tau(G)$ of G. These notions are similar to that used in the previous section.

We also consider a relaxed version of the above notions, where a vertex u from $V(G) \setminus S$ belongs to the interval of S if $N_G(u) \cap S$ *contains* some set from $\tau(u)$ instead of requiring to be equal to some such a set. In other words, a way of introducing this variant is by suitably modifying the function τ. Therefore, let the *closure* $\bar{\tau}$ of τ be the threshold function

$$\bar{\tau} : V(G) \to 2^{V(G)} : u \mapsto \left\{\bar{N} : \bar{N} \subseteq N_G(u) \text{ and } \exists N \in \tau(u) : N \subseteq \bar{N}\right\}.$$

The sets $I_{\bar{\tau}}(S)$ and $H_{\bar{\tau}}(S)$ are the *relaxed τ-interval of S*, and the *relaxed τ-hull of S*, respectively. The $\bar{\tau}$-threshold process on G starting with S is the *relaxed τ-threshold process on G starting with S*. The *relaxed τ-interval number* of G is $i_{\bar{\tau}}(G)$, and the *relaxed τ-hull number* of G is $h_{\bar{\tau}}(G)$.

In the example given by Figure 3.3, considering $\bar{\tau} = \tau$ and $S = \{v_3\}$, we can see that $I_{\bar{\tau}}(S) = \{v_2, v_3\}$, since $\bar{\tau}(v_2)$ contains $\{v_1, v_3\}$.

As seen in Chapter 1, special choices for τ lead to many well known graph parameters. Those examples already imply several hardness results.

In view of algorithmic aspects though, some remarks concerning encoding lengths are necessary. Given a graph G of order $n(G)$, and a threshold function τ on G, we can encode the pair (G, τ) listing, for every vertex u of G, the $d_G(u)$ neighbors of u by indicating their names, which are 0/1-vectors of length $O(\ln n(G))$, as well as the $|\tau(u)|$ distinct 0/1-incidence vectors of length $d_G(u)$ of the sets in $\tau(u)$, where the i-th entry of such a vector corresponds to the neighbor of u with the i-th smallest name. This encoding has size

$$O\left(\sum_{u \in V(G)} \Big(d_G(u)\Big(\ln(n(G)) + |\tau(u)|\Big)\Big)\right). \tag{3.4}$$

In the remainder of this section we assume the input is encoded as above. Note that $|\tau(u)|$ can be $\Omega\left(2^{d_G(u)}\right)$, hence this size is not necessarily polynomial in $n(G)$. However, we assume that the hardest instances have size $O(poly(n(G)))$.

An interesting threshold function based on `xor`-gates of logic circuits is the following parity related function:

$$\oplus : V(G) \to 2^{V(G)} : u \mapsto \left\{N \in 2^{N_G(u)} : |N| \mod 2 = 1\right\}.$$

Clearly, the reasonable encoding length of the pair $(G, \oplus)$ is just the encoding length of G, which is much smaller than the expression in (3.4). Note that $\oplus$-hull number of a tree equals 1.

3.3.1 Hardness results

Our first hardness results concern complete graphs, that is, even on extremely simple graphs sufficiently complex threshold functions lead to hard problems.

Theorem 3.22 *Given a pair (k, τ), where k is a non-negative integer and τ is a threshold function on a complete graph K_n, it is NP-complete to decide whether $i_{\bar{\tau}}(K_n) \leq k$ and it is NP-complete to decide whether $h_{\bar{\tau}}(K_n) \leq k$.*

Proof. The two considered decision problems are clearly in NP, because the $\bar{\tau}$-threshold process on G starting with a given set can be generated in polynomial time. In order to establish hardness, we describe polynomial reductions from Set Cover. Therefore, let $\mathcal{C} = (k, (S_1, \ldots, S_p))$ be an instance of Set Cover, that is,

k is a positive integer, the S_i are subsets of some ground set V, and $\mathcal{C}$ is a 'Yes'-instance of Set Cover if and only if there is a set C of at most k integers in $[p]$ with $\bigcup_{i \in C} S_i = V$. Clearly, we may assume that $\bigcup_{i \in [p]} S_i = V$ and $k < \min\{p, |V|\}$.

Let $n = p + |V|$, and let $V(K_n) = \{S_1, \ldots, S_p\} \cup V$. For $i \in [p]$, let $\tau(S_i) = \{\emptyset\}$, and, for $u \in V$, let $\tau(u) = \{\{S_i\} : i \in [p] \text{ and } u \in S_i\}$. It is easy to see that $\mathcal{C}$ is a 'Yes'-instance of Set Cover if and only if $i_{\bar{\tau}}(K_n) \leq k$. Since the encoding length of the Set Cover instance $\mathcal{C}$ is $\Omega(\ln(k) + p|V|)$, the special choice of τ implies that the encoding length of (K_n, τ) is polynomially bounded in terms of the encoding length of $\mathcal{C}$.

Let $n' = 2p + 1 + |V|$, and let $V(K_{n'}) = \{S_1, \ldots, S_p\} \cup V \cup X$, where X is a set of order $p + 1$ that is disjoint from $\{S_1, \ldots, S_p\} \cup V$. For $i \in [p]$, let $\tau'(S_i) = \{X\}$, for $u \in V$, let $\tau'(u) = \{\{S_i\} : i \in [p] \text{ and } u \in S_i\}$, and, for $x \in X$, let $\tau'(x) = \{V\}$. Again, the encoding length of $(K_{n'}, \tau')$ is polynomially bounded in terms of the encoding length of $\mathcal{C}$.

Suppose that $\mathcal{C}$ is a 'Yes'-instance of Set Cover, and let C be as above. Let $S = \{S_i : i \in C\}$. The definition of τ' implies that $I_{\bar{\tau}'}(S) = S \cup V$, $I^2_{\bar{\tau}'}(S) = S \cup V \cup X$, and $I^3_{\bar{\tau}'}(S) = V(K_{n'})$, which implies $h_{\bar{\tau}'}(K_{n'}) \leq k$.

Conversely, suppose that $h_{\bar{\tau}'}(K_{n'}) \leq k$. Let S be a set of at most k vertices of $K_{n'}$ with $H_{\bar{\tau}'}(S) = V(K_{n'})$ such that $|S \cap V|$ is as small as possible. If $u \in S \cap V$ and S_i is such that $u \in S_i$, then $u \in I_{\bar{\tau}'}((S \setminus \{u\}) \cup \{S_i\})$, which implies $H_{\bar{\tau}'}((S \setminus \{u\}) \cup \{S_i\}) = V(K_{n'})$. Since $|(S \setminus \{u\}) \cup \{S_i\}| \leq |S|$ and $|((S \setminus \{u\}) \cup \{S_i\}) \cap V| < |S \cap V|$, this yields a contradiction to the choice of S. Hence S contains no element of V. Since $k < p$, the set S does not contain all elements of X. If $V \not\subseteq I_{\bar{\tau}'}(S)$, then the definition of τ' implies $I_{\bar{\tau}'}(S) = H_{\bar{\tau}'}(S) \neq V(K_{n'})$, which is a contradiction. Hence, $V \subseteq I_{\bar{\tau}'}(S)$. We obtain $\bigcup_{S_i \in S} S_i = V$, which yields that $\mathcal{C}$ is a 'Yes'-instance of Set Cover. $\square$

As observed by Chlebík and Chlebíková [37], Feige [70], Dinur and Steurer [55] showed the hardness of approximation of Set Cover for instance with $\ln(p+|V|) \approx \ln(|V|)$. Therefore, the construction in the proof of Theorem 3.22 actually implies the following.

Theorem 3.23 *For a given threshold function τ on a complete graph K_n, neither $i_{\bar{\tau}}(K_n)$ nor $h_{\bar{\tau}}(K_n)$ can be approximated in polynomial time within a factor of $(1 - \epsilon)\ln(n)$ for any constant $\epsilon > 0$ unless $P = NP$.*

Replacing Set Cover with Exact Cover by 3-Sets (cf. the proof of Theorem 3.26 below) within the proof of Theorem 3.22 yields the following additional result.

Theorem 3.24 *Given a pair (k,τ), where k is a non-negative integer and τ is a threshold function on a complete graph K_n, it is NP-complete to decide whether $i_\tau(K_n) \leq k$.*

Note that in the construction used in the proof of Theorem 3.22 several edges of the complete graph K_n are in fact irrelevant because of the special choice of τ. Removing these edges easily implies that deciding $i_{\bar{\tau}}(G) \leq k$ and $i_\tau(G) \leq k$ are NP-complete for a given triple (G, k, τ), where G is a bipartite graph, k is a non-negative integer, and τ is a threshold function on G.

We present some more hardness results concerning special choices of the threshold function.

Theorem 3.25 *Given a graph G and a given integer k, it is NP-complete to decide whether $i_\tau(G) \leq k$, where τ is the threshold function on G with $\tau(u) = \binom{N_G(u)}{1}$ for every vertex u of G.*

Proof. Again, the considered decision problem is clearly in NP. In order to prove completeness, we describe a polynomial reduction from 1-IN-3SAT. Therefore, let $\mathcal{F}$ be an instance of 1-IN-3SAT with m clauses $C_1, \ldots, C_m$ over n Boolean variables $x_1, \ldots, x_n$. We specify a graph G and an integer k such that the order of G is polynomially bounded in terms of n and m, and $\mathcal{F}$ has a satisfying truth assignment that leads to exactly one true literal in each clause if and only if $i_\tau(G) \leq k$ for the special threshold function τ described in the statement.

Therefore,

- for every variable x_i, we create a copy G_i of $K_4 - e$ and denote the two vertices of degree 3 in G_i by x_i and $\bar{x}_i$,
- for every clause C_j, we create a vertex C_j, and
- for every literal $x \in \{x_1, \ldots, x_n\} \cup \{\bar{x}_1, \ldots, \bar{x}_n\}$ and every clause C_j such that x appears in C_j, we add the edge xC_j.

This completes the construction of G. Let $k = n$.

If $\mathcal{F}$ has a satisfying truth assignment that leads to exactly one true literal in each clause, then the set S of all vertices that correspond to true literals is a τ-interval set of order n. Conversely, if S is a τ-interval set of order at most n, then the special choice of τ and k implies that $S \cap V(G_i) \in \{\{x_i\}, \{\bar{x}_i\}\}$ for every $i \in [n]$, and $S \cap \{C_1, \ldots, C_m\} = \emptyset$. Therefore, since S is a τ-interval set, the elements in S indicate a satisfying truth assignment for $\mathcal{F}$ that leads to exactly one true literal in each clause, which completes the proof. □

Theorem 3.26 *Given a bipartite graph G and a given integer k, it is NP-complete to decide whether $i_\oplus(G) \leq k$.*

Proof. Again, the considered decision problem is clearly in NP. In order to prove completeness, we describe a polynomial reduction from Exact Cover by 3-Sets. Therefore, let $\mathcal{C}$ be an instance of Exact Cover by 3-Sets consisting of m 3-element subsets $C_1, \ldots, C_m$ of a ground set X of order $3n$. Clearly, we may assume that $\bigcup_{i=1}^m C_i = X$. We specify a bipartite graph G and an integer k such that the order of G is polynomially bounded in terms of n and m, and $\mathcal{C}$ is a 'Yes'-instance of Exact Cover by 3-Sets if and only if $i_\oplus(G) \leq k$. Let $V(G) = X \cup \{C_1, \ldots, C_m\} \cup \{r\} \cup \{s_1, \ldots, s_{n+2}\}$,

$$\begin{aligned} E(G) &= \{xC_i : x \in X, i \in [m], \text{ and } x \in C_i\} \cup \{rC_i : i \in [m]\} \cup \{rs_j : j \in [n+2]\}, \\ &\text{and} \\ k &= n+1. \end{aligned}$$

If $\mathcal{C}$ is a 'Yes'-instance, and I is a set of exactly n indices in $[m]$ such that $\bigcup_{i \in I} C_i = X$, then $\{r\} \cup \{C_i : i \in I\}$ is an $\oplus$-interval set of G, which implies $i_\oplus(G) \leq n+1$. Conversely, let S be an $\oplus$-interval set of G of order at most $n+1$. If S does not contain r, then $s_j \in S$ for every $j \in [n+2]$, which implies the contradiction $|S| > n+1$. Hence, $r \in S$. Let $n_X = |S \cap X|$ and $n_C = |S \cap \{C_1, \ldots, C_m\}|$. Since each of the $3n - n_X$ vertices in $X \setminus S$ has at least one neighbor in $S \cap \{C_1, \ldots, C_m\}$, and every vertex C_i has exactly 3 neighbors in X, we obtain $3n - n_X \leq 3n_C$, which implies $\frac{1}{3}n_X + n_C \geq n$. Since $n_X + n_C \leq |S| - 1 \leq n$, we obtain $n_X = 0$. Therefore, if $I = \{i \in [m] : C_i \in S\}$, then $|I| = n_C \leq n$ and $\bigcup_{i \in I} C_i = X$, which implies that $\mathcal{C}$ is a 'Yes'-instance. □

3.3.2 Efficient algorithms for trees

In view of the strong hardness results in the previous subsection, efficient algorithms seem possible only for either quite restricted graph classes (and general threshold functions) or quite restricted threshold functions (and more general graph classes). The second of these two options comprises the many well known efficient algorithms for domination, multiple domination, vertex cover, independence and target set selection in special graph classes. Therefore, we focus here on the first option, that is, we do not want to restrict the threshold functions.

In [40, 41] generalizations of *target selection* problem (a special case of the problem studied here) have been studied on trees where size/cost set bound and time bound are considered. They provided linear algorithms for such cases. However these problems assume that any vertex v has an integer value $t(v)$ meaning that v

needs $t(v)$ activate neighbors to become activate. Clearly relaxed τ-threshold process provides more general problems such as relaxed τ-interval (time bound equals 1) and relaxed τ-hull (time bound equals $n(G)$).

Relaxed and non-relaxed τ-hull number

We present some efficient algorithms for trees. Our first result concerns the relaxed τ-hull number.

Lemma 3.27 *Let G be a graph, and let τ be a threshold function on G. Let uv be an edge of G such that v has degree 1 in G.*

(i) If $\tau(v) = \emptyset$, then for the threshold function σ on $G - v$ defined as

$$\sigma(x) = \begin{cases} \{N \setminus \{v\} : N \in \tau(u)\} & \text{, if } x = u \text{, and} \\ \tau(x) & \text{, otherwise,} \end{cases}$$

it holds that $h_{\bar{\tau}}(G) = h_{\bar{\sigma}}(G - v) + 1$.

(ii) If $\emptyset \in \tau(v)$, and the threshold function σ on $G - v$ is as in (i), then $h_{\bar{\tau}}(G) = h_{\bar{\sigma}}(G - v)$.

(iii) If $\tau(v) = \{\{u\}\}$, and the threshold function σ on $G - v$ is such that

$$\sigma(x) = \begin{cases} \{N : N \in \tau(u) \text{ and } v \notin N\} & \text{, if } x = u \text{, and} \\ \tau(x) & \text{, otherwise,} \end{cases}$$

then $h_{\bar{\tau}}(G) = h_{\bar{\sigma}}(G - v)$.

Proof. (i) Since $\bar{\tau}(v) = \emptyset$, every $\bar{\tau}$-hull set of G contains v. The definition of σ easily implies that S is a $\bar{\tau}$-hull set of G if and only if $S \setminus \{v\}$ is a $\bar{\sigma}$-hull set of $G - v$, which implies (i).

(ii) Since $\emptyset \in \bar{\tau}(v)$, no $\bar{\tau}$-hull set of G that is minimal with respect to inclusion contains v. For a set $S \subseteq V(G) \setminus \{v\}$, the definition of σ easily implies that S is a $\bar{\tau}$-hull set of G if and only if S is a $\bar{\sigma}$-hull set of $G - v$, which implies (ii).

(iii) If S is a $\bar{\tau}$-hull set of G that is minimal with respect to inclusion and $v \in S$, then $u \notin S$, and $(S \setminus \{v\}) \cup \{u\}$ is a $\bar{\tau}$-hull set of G. Hence, G has a $\bar{\tau}$-hull set of minimum order that does not contain v. For a set $S \subseteq V(G) \setminus \{v\}$, the definition of σ easily implies that S is a $\bar{\tau}$-hull set of G if and only if S is a $\bar{\sigma}$-hull set of $G - v$, which implies (iii). $\square$

Iteratively applying Lemma 3.27 immediately implies the following.

Theorem 3.28 *For a given pair (T, τ), where T is a tree and τ is a threshold function on T, the relaxed τ-hull number of T can be determined in linear time.*

Note that the linear running time refers to the encoding length as specified in (3.4).

The non-relaxed τ-hull number seems to be much harder even for trees. More specifically, given a tree T rooted by some vertex u and a threshold function τ on T, it seems difficult to verify even if u does not belong to any τ-hull set S of T. This occurs because u needs an exact subset $N \in \tau(u)$ of its neighbors at some time step $t \geq 1$ during the process, such that $N \subseteq H(S)$ at t and all of the vertices in $N_G(u) \setminus N$ are not in $H(S)$ at t. Hence, it is required a mutual synchronization of the τ-hull sets of all subtrees of u. However, a simple dynamic programming approach works for paths.

Proposition 3.29 *For a given pair (P_n, τ), where τ is a threshold function on the path P_n of order n, the τ-hull number of P_n can be determined in polynomial time.*

Proof. For every two vertices u and v of P_n, it is possible to determine efficiently whether $H_\tau(\{u, v\})$ contains all vertices that lie between u and v on P_n. Similarly, for every vertex u of P_n, it is possible to determine efficiently whether $H_\tau(\{u\})$ contains all vertices that lie between u and some specific endvertex of P_n. In view of these observations, a simple dynamic programming approach allows to determine the τ-hull number of P_n efficiently. $\square$

Relaxed and non-relaxed τ-interval number

Remember that we cannot able to answer whether a given vertex does not belong to a τ-hull set of a tree T due to the requirement of a synchronization of the τ-hull sets of the subtrees of a vertex. The next two results concern the recognition of vertices that are not in every τ-interval set for trees.

Lemma 3.30 *Let T be a tree, and let τ be a threshold function on T. Let u be a vertex. For $v \in N_T(u)$, let T_v be the component of $T - u$ that contains v.*

There is a τ-interval set S of T that does not contain u if and only if there is some set N in $\tau(u)$ such that for every $v \in N_G(u) \setminus N$, there is a τ_v-interval set S_v of T_v that does not contain v, where

$$\tau_v(x) = \begin{cases} \{X : X \in \tau(v) \text{ and } u \notin X\} & \text{, if } x = v, \text{ and} \\ \tau(x) & \text{, } x \in V(T_v) \setminus \{v\}. \end{cases}$$

Proof. Suppose that S is a τ-interval set of T that does not contain u. Clearly, $N = S \cap N_G(u) \in \tau(u)$, and if $v \in N_G(u) \setminus N$, then $S \cap V(T_v)$ is a τ_v-interval set of T_v that does not contain v. This proves the necessity.

For the sufficiency, let $N \in \tau(u)$, and S_v for $v \in N_G(u)\setminus N$ be as in the statement. By the definition of τ_v, the set $S = \bigcup_{v\in N} V(T_v) \cup \bigcup_{v\in N_G(u)\setminus N} S_v$ is a τ-interval set of T that does not contain u. $\square$

Proposition 3.31 *For a given triple (T, τ, u), where T is a tree, τ is a threshold function on T, and u is a vertex of T, one can determine in linear time whether T has a τ-interval set that does not contain u.*

Proof. We root T in u. For every vertex v of T that is distinct from u, let T_v denote the subtree of T that contains v as well as all descendants of v, and that is rooted in v. Furthermore, if v^- is the parent of v, then let τ_v be the threshold function on T_v defined by

$$\tau_v(x) = \begin{cases} \{X : X \in \tau(v) \text{ and } v^- \notin X\} & \text{, if } x = v, \text{ and} \\ \tau(x) & \text{, } x \in V(T_v) \setminus \{v\}. \end{cases}$$

Iteratively applying Lemma 3.30 and processing the vertices v of T in an order of non-increasing depth, one can determine in linear time, whether T_v has a τ_v-interval set that does not contain v. Note that if exists $N \in \tau(u)$ such that for every $v \in N_G(u) \setminus N$, T_v has a τ-interval set that does not contain v, then T has a τ-interval set S that does not contain u, because, without loss of generality, $\bigcup_{w\in N} V(T_w) \subseteq S$. $\square$

Our next goal is an efficient algorithm for the τ-interval number of a tree.

Lemma 3.32 *Let T be a rooted tree, and let τ be a threshold function on T. For every vertex u of T that is not a leaf, every child v of u, and every subset F of $\{u, v\}$, let*

$$i(u, v, F) = \min\Big\{|S \setminus \{u\}| : S \textit{ is a } \tau_{uv}\textit{-interval set of } T_{uv} \textit{ with } S \cap \{u, v\} = F\Big\},$$

where T_{uv} is the subtree of T that contains u, v, and all descendants of v, τ_{uv} is the threshold function on T_{uv} such that

$$\tau_{uv}(x) = \begin{cases} \{F \cap \{v\}\} & \textit{, if } x = u, \textit{ and} \\ \tau(x) & \textit{, } x \in V(T_{uv}) \setminus \{u\}, \end{cases}$$

and $\min \emptyset = \infty$.

(i) If v is a leaf, then

$$i(u,v,F)=\begin{cases}1 & \text{, if } v\in F,\\ 0 & \text{, if } v\not\in F \text{ and } F\cap\{u\}\in\tau(v)\text{, and}\\ \infty & \text{, otherwise.}\end{cases}$$

(ii) If v is not a leaf, W is the set of children of v, and $v\not\in F$, then

$$i(u,v,F)=\min\Bigg\{\sum_{w\in W\cap N} i(v,w,\{w\})+\sum_{w\in W\setminus N} i(v,w,\emptyset) : N\in\tau(v) \text{ with } N\cap\{u\}=F\cap\{u\}\Bigg\}.$$

(iii) If v is not a leaf, W is the set of children of v, and $v\in F$, then

$$i(u,v,F)=1+\sum_{w\in W}\min\Big\{i(v,w,\{v\}),i(v,w,\{v,w\})\Big\}.$$

Proof. (i) Note that $V(T_{uv})=\{u,v\}$. The set F is a τ_{uv}-interval set of T_{uv} if and only if either $v\in F$ or $v\not\in F$ but $F\cap\{u\}\in\tau(v)$. By the definition of $i(u,v,F)$, this immediately yields the stated values.

(ii) Let S be a τ_{uv}-interval set of T_{uv} with $S\cap\{u,v\}=F$ and $i(u,v,F)=|S\setminus\{u\}|$. Since $v\not\in F$, we obtain that $v\not\in S$. Hence, the set $N=S\cap N_{T_{uv}}(v)$ belongs to $\tau(v)$ and satisfies $N\cap\{u\}=F\cap\{u\}$. For every child w of v, let $S_w=S\cap V(T_{vw})$. If $w\in W\cap N$, then S_w is a τ_{vw}-interval set of T_{vw} with $S_w\cap\{v,w\}=\{w\}$, and, if $w\in W\setminus N$, then S_w is a τ_{vw}-interval set of T_{vw} with $S_w\cap\{v,w\}=\emptyset$. This implies

$$\begin{aligned} i(u,v,F) &= |S\setminus\{u\}|=\sum_{w\in W\cap N}|S_w|+\sum_{w\in W\setminus N}|S_w|\\ &\geq \sum_{w\in W\cap N} i(v,w,\{w\})+\sum_{w\in W\setminus N} i(v,w,\emptyset),\end{aligned}$$

which implies that $i(u,v,F)$ is at least the minimum stated in (ii).

Conversely, let $N\in\tau(v)$ with $N\cap\{u\}=F\cap\{u\}$ be such that

$$\sum_{w\in W\cap N} i(v,w,\{w\})+\sum_{w\in W\setminus N} i(v,w,\emptyset)$$

is minimum. For $w\in W\cap N$, let S_w be a τ_{vw}-interval set of T_{vw} with $S_w\cap\{v,w\}=\{w\}$ and $i(v,w,\{w\})=|S_w\setminus\{v\}|$, and, for $w\in W\setminus N$, let S_w be a τ_{vw}-interval set of T_{vw} with $S_w\cap\{v,w\}=\emptyset$ and $i(v,w,\emptyset)=|S_w\setminus\{v\}|$. By the definition of τ_{uv} and the choice of N, the set $S = F \cup \bigcup_{w \in W} S_w$ is a τ_{uv}-interval set of T_{uv} with $S \cap \{u, v\} = F$. We obtain

$$\begin{aligned} i(u, v, F) &\leq |S \setminus \{u\}| = \sum_{w \in W \cap N} |S_w| + \sum_{w \in W \setminus N} |S_w| \\ &= \sum_{w \in W \cap N} i(v, w, \{w\}) + \sum_{w \in W \setminus N} i(v, w, \emptyset), \end{aligned}$$

which implies that $i(u, v, F)$ is at most the minimum stated in (ii).

(iii) Let S be a τ_{uv}-interval set of T_{uv} with $S \cap \{u, v\} = F$ and $i(u, v, F) = |S \setminus \{u\}|$. For every child w of v, let $S_w = S \cap V(T_{vw})$. Since $v \in F \subseteq S$, the set S_w is a τ_{vw}-interval set of T_{vw} with $v \in S_w$. We obtain

$$\begin{aligned} i(u, v, F) &= |S \setminus \{u\}| = |\{v\}| + \sum_{w \in W} |S_w \setminus \{v\}| \\ &\geq 1 + \sum_{w \in W} \min \Big\{ i(v, w, \{v\}), i(v, w, \{v, w\}) \Big\}, \end{aligned}$$

which implies that $i(u, v, F)$ is at least the minimum stated in (iii).

Conversely, for $w \in W$, let S_w be a τ_{vw}-interval set of T_{vw} with $v \in S_w$ and

$$\min \Big\{ i(v, w, \{v\}), i(v, w, \{v, w\}) \Big\} = |S_w \setminus \{v\}|.$$

By the definition of τ_{uv}, the set $S = F \cup \bigcup_{w \in W} S_w$ is a τ_{uv}-interval set of T_{uv} with $S \cap \{u, v\} = F$. We obtain

$$\begin{aligned} i(u, v, F) &\leq |S \setminus \{u\}| = |\{v\}| + \sum_{w \in W} |S_w \setminus \{v\}| \\ &= 1 + \sum_{w \in W} \min \Big\{ i(v, w, \{v\}), i(v, w, \{v, w\}) \Big\}, \end{aligned}$$

which implies that $i(u, v, F)$ is at most the minimum stated in (iii). $\square$

Theorem 3.33 *For a given pair (T, τ), where T is a tree of order n and τ is a threshold function on T, the τ-interval number of T can be determined in linear time.*

Proof. We root T in an endvertex r. Let s be the neighbor of r in T. By Lemma 3.32, the vectors $\Big(i(u, v, \emptyset), i(u, v, \{u\}), i(u, v, \{v\}), i(u, v, \{u, v\}) \Big)$ can be determined in linear time processing the edges uv of T in an order of non-increasing depth of u.

By definition,

$$i_\tau(T) = \min\Big\{ i(r,s,F) + |F \cap \{r\}| : F \subseteq \{r,s\} \text{ with either } r \in F \text{ or } r \notin F \text{ and } F \cap \{s\} \in \tau(r) \Big\}.$$

□

By definition, the relaxed τ-interval number of a tree T equals the $\bar{\tau}$-interval number of T, and we can apply the algorithm described in the proof of Theorem 4.12. Unfortunately, this only leads to a running time that is linear in the encoding length of $(T, \bar{\tau})$, which may be much bigger than the encoding length of (T, τ). Therefore, for the next result, we need to argue how to obtain a running time that is linear in the encoding length of (T, τ).

Corollary 3.34 *For a given pair (T, τ), where T is a tree of order n and τ is a threshold function on T, the relaxed τ-interval number of T can be determined in linear time.*

Proof. In view of Lemma 3.32, it suffices to argue, how to evaluate the expression

$$\min\left\{ \sum_{w \in W \cap \bar{N}} i(v,w,\{w\}) + \sum_{w \in W \setminus \bar{N}} i(v,w,\emptyset) : \bar{N} \in \bar{\tau}(v) \text{ with } \bar{N} \cap \{u\} = F \cap \{u\} \right\}$$

in $O(d_G(v)|\tau(v)|)$ time, where v is a child of u, and $v \notin F \subseteq \{u,v\}$. By the definition of $\bar{\tau}$, we obtain that $\bar{N} \in \bar{\tau}(v)$ if and only if there is a set $N \in \tau(v)$ with $N \subseteq \bar{N}$, which easily implies that the above expression equals

$$\min\Big\{ \sum_{w \in W \cap N} i(v,w,\{w\}) + \sum_{w \in W \setminus N} \min\{i(v,w,\emptyset), i(v,w,\{w\})\} : N \in \tau(v) \text{ with } N \cap \{u\} \subseteq F \cap \{u\} \Big\},$$

which can clearly be determined in $O(d_G(v)|\tau(v)|)$ time. □

The problem of computing a smallest $\oplus$-interval set is close related to parity domination problem [79], which can be solved in linear time for graphs with bounded treewidth [79]. However, parity domination algorithms cannot be applied to obtain the $\oplus$-interval number of a graph, because the parity domination set property depends on a given partition of the vertices and a given function $\pi : V \to \{0,1\}$. For more information about parity domination see [53, 79, 88].

Hence, in order to determine the $\oplus$-interval number in linear time, the special choice $\tau = \oplus$ requires some arguments concerning the running time.

Theorem 3.35 *For a given tree T, the $\oplus$-interval number of T can be determined in linear time.*

Proof. Again, we can root T in an endvertex r and apply the algorithm described in the proof of Theorem 4.12. Now, we only need to argue how to improve its running time to be linear in the encoding length of T. Consider w^- the parent of a vertex w of T. In view of Lemma 3.32, it suffices to explain how to minimize an expression of the form

$$\sum_{w \in N} i_1(w) + \sum_{w \in W \setminus N} i_2(w)$$

over all subsets N of a set W of a fixed parity p modulo 2 in $O(|W|)$ time, where

$$i_1(w) = \left| \sum_{s \in N(w) \setminus \{w^-\}} \min \Big\{ i(w, s, F) : F \subseteq \{r, s\} \text{ and } r \in F \Big\} \right| + 1,$$

and

$$i_2(w) = \left| \sum_{s \in N(w) \setminus \{w^-\}} \min \Big\{ i(w, s, F) : F \subseteq \{r, s\} \text{ and } r \notin F \Big\} \right|.$$

If $i_1(w) = i_2(w)$ for some $w \in W$, then the minimum equals $\sum_{w \in W} \min\{i_1(w), i_2(w)\}$. Hence, we may assume that $i_1(w) \neq i_2(w)$ for every $w \in W$. Let $N = \{w \in W : i_1(w) < i_2(w)\}$. If $|N| \mod 2 = p$, then the minimum equals again $\sum_{w \in W} \min\{i_1(w), i_2(w)\}$. If $|N| \mod 2 \neq p$, and $w^* \in W$ is such that

$$|i_1(w^*) - i_2(w^*)| = \min\{|i_1(w) - i_2(w)| : w \in W\},$$

then the minimum equals $|i_1(w^*) - i_2(w^*)| + \sum_{w \in W} \min\{i_1(w), i_2(w)\}$. These observations easily imply the linear running time. $\square$

Chapter 4

Editing Problems by the Removal of a Matching

"If I look back I am lost."
(Daenerys Targarean)

— G. Martin, *A Dance with Dragons*

As a natural variant of the many decycling notions studied in graphs, we consider two problems to decide whether a given graph G has a matching M such that $G - M$ is a forest, and whether G is a bipartite graph, respectively.

Concerning the first problem, we establish NP-completeness for 2-connected planar subcubic graphs, and describe polynomial time algorithms that also determine such a matching if it exists for graphs that are claw- and paw-free, P_5-free graphs, chordal graphs, and C_4-free distance hereditary graphs.

In order to obtain a bipartite graph by the removal of a matching, we show that such a decision problem is NP-complete even for planar graphs of maximum degree 4, but can be solved in linear time in graphs of maximum degree 3. We also present polynomial time algorithms for (claw, paw)-free graphs, graphs containing only triangles as odd cycles, graphs with bounded dominating sets, and P_5-free graphs. In addition, we show that the problem is fixed-parameter tractable when parameterized by clique-width, which implies polynomial time solvability for many interesting graph classes such as distance-hereditary graphs and outerplanar graphs. Finally a $2^{\beta(G)}.n$ algorithm, and a kernel having at most $2.nd(G)$ vertices are presented, where $nd(G)$ is the neighborhood diversity of the input graph.

4.1 Obtaining a Forest

For a set E of edges of a graph G, let $G - E$ be the graph with vertex set $V(G)$ and edge set $E(G) \setminus E$. If $G - E$ is a forest, then E is *decycling*. Let $\mathcal{FM}$ be the set of all graphs that have a decycling matching.

The following lemma collects some basic observations concerning graphs that have a decycling matching.

Lemma 4.1 *Let G be a graph.*

(i) If $G \in \mathcal{FM}$ is connected, then G has a matching M for which $G - M$ is a tree.

(ii) If $G \in \mathcal{FM}$, then $m(H) \leq \left\lfloor \frac{3n(H)}{2} \right\rfloor - 1$ for every subgraph H of G.

(iii) If G is subcubic and connected, then $G \in \mathcal{FM}$ if and only if G has a spanning tree T such that all endvertices of T are of degree at most 2 in G.

Proof. (i) Let $G \in \mathcal{FM}$ be connected. Let M be a matching of G such that $G - M$ is a forest F with as few components as possible. Suppose, for a contradiction, that F is not connected. Since G is connected, M contains an edge e between different components of F. Now, $N = M \setminus \{e\}$ is a matching of G such that $G - N$ is a forest with less components than F, which implies a contradiction. Hence, F is a tree.

(ii) Let $G \in \mathcal{FM}$. Since $\mathcal{FM}$ is closed under taking subgraphs, it suffices to show $m(G) \leq \left\lfloor \frac{3n(G)}{2} \right\rfloor - 1$. Let M be a decycling matching of G. Clearly, $m(G) \leq m(G - M) + |M| \leq (n(G) - 1) + \left\lfloor \frac{n(G)}{2} \right\rfloor = \left\lfloor \frac{3n(G)}{2} \right\rfloor - 1$.

(iii) Let G be a connected subcubic graph. Clearly, we may assume that $n(G) \geq 3$.

First, suppose that $G \in \mathcal{FM}$. By (i), G has a matching M such that $G - M$ is a spanning tree T. If u is an endvertex of T, then $d_G(u) \leq d_T(u) + 1 \leq 2$, which implies that all endvertices of T have degree at most 2 in G.

Next, suppose that T is a spanning tree of G such that all endvertices of T are of degree at most 2 in G. Let $M = E(G) \setminus E(T)$. Clearly, M is decycling, and it remains to show that M is a matching. Suppose that M contains two edges incident with the same vertex u of G. This implies $d_T(u) \leq d_G(u) - 2 \leq 3 - 2 = 1$, that is, u is an endvertex of T. By the choice of T, we obtain $d_T(u) \leq d_G(u) - 2 \leq 2 - 2 = 0$, which is a contradiction. □

Lemma 4.1(iii) is the key observation for the following hardness result.

Theorem 4.2 *For a given 2-connected planar subcubic graph G, it is NP-complete to decide whether $G \in \mathcal{FM}$.*

Proof. The considered decision problem is clearly in NP. In order to show NP-completeness, we use [77] that deciding the existence of a Hamiltonian cycle for a given 3-connected planar cubic graph is NP-complete. In fact, the 3-connected planar cubic graphs G constructed by Garey et al. in [77] contain several edges that necessarily belong to every Hamiltonian cycle of G; regardless of whether such a cycle exists or not. Therefore, removing such an edge, their construction implies the NP-completeness of the following decision problem: *Given a 2-connected planar subcubic graph G with exactly two vertices u and v of degree 2, does G have a Hamiltonian path whose endvertices are u and v?*

Let G be a 2-connected planar subcubic graph with exactly two vertices u and v of degree 2. In order to complete the proof, it suffices to show that G has a Hamiltonian path whose endvertices are u and v if and only if $G \in \mathcal{FM}$. First, suppose that P is a Hamiltonian path of G whose endvertices are u and v. Clearly, P is a spanning tree of G such that all endvertices of P are of degree at most 2 in G. By Lemma 4.1(iii), this implies $G \in \mathcal{FM}$. Next, suppose that $G \in \mathcal{FM}$. By Lemma 4.1(iii), this implies that G has a spanning tree T such that all endvertices of T are of degree at most 2 in G. Since u and v are the only vertices of G of degree at most 2, this implies that T has exactly the two endvertices u and v. Hence, T is a Hamiltonian path of G whose endvertices are u and v. □

In order to enable suitable reductions, we now consider a slightly more general version of our decision problem.

Allowed Decycling Matching
Input: A graph G and a set F of edges of G.
Question: Is G has a decycling matching M that does not intersect F, and determine such a matching if it exists

A matching M as in Allowed Decycling Matching is an *allowed decycling matching* of (G, F).

Theorem 4.3 Allowed Decycling Matching *can be solved in polynomial time for (claw, paw)-free graphs.*

Proof. Let G be a (claw, paw)-free graph and let F be a set of edges of G. Since (G, F) has an allowed decycling matching if and only if $(K, E(K) \cap F)$ has an allowed decycling matching for every component K of G, we may assume that G is connected.

The following claim is an immediate consequence of Lemma 4.1(ii).

Claim 4.4 *If G contains K_4 as an induced subgraph, then (G, F) has no allowed decycling matching.*

Claim 4.5 *If G has a vertex of degree at least 4, then (G, F) has no allowed decycling matching.*

Proof of Claim 4.5: Let u be a vertex of G with four neighbors v_1, v_2, v_3, and v_4. Since G is $\{\text{claw}, \text{paw}, K_4\}$-free, we may assume, by symmetry, that $v_1v_2, v_2v_3 \in E(G)$ and $v_1v_3 \notin E(G)$. Considering v_1, v_3, and v_4, this implies, by symmetry, that $v_3v_4 \in E(G)$. Considering the three triangles uv_1v_2u, uv_2v_3u, and uv_3v_4u, it follows that (G, F) does not have an allowed decycling matching. $\square$

Since no endvertex of G lies on a cycle, we may assume that G has minimum degree at least 2. Since whether G is K_4-free and has maximum degree at most 3, can be tested in polynomial time, we may assume, by Claim 4.4 and Claim 4.5, that G is K_4-free and has maximum degree at most 3. If G does not have any vertex of degree 3, then G is a cycle, and (G, F) has an allowed decycling matching if and only if F does not contain all edges of G. Hence, we may assume that G has a vertex b of degree 3. Let $N_G(b) = \{a, c, d\}$. Since G is $\{\text{claw}, \text{paw}, K_4\}$-free, we may assume, by symmetry, that $ac, cd \in E(G)$ and $ad \notin E(G)$. Let $G' = (V(G) \setminus \{b, c\}, (E(G) \setminus \{ab, ac, bc, bd, cd\}) \cup \{ad\})$, and let $F' = (F \setminus \{ab, ac, bc, bd, cd\}) \cup \{ad\} \cup \{xa : x \in N_G(a) \setminus \{b, c\}\} \cup \{ad\} \cup \{yd : x \in N_G(d) \setminus \{b, c\}\}$.

Claim 4.6 *(i) G' is (claw, paw)-free.*

(ii) (G, F) has an allowed decycling matching if and only if

- *$ab, cd \notin F$ or $ac, bd \notin F$, and*
- *(G', F') has an allowed decycling matching.*

Proof of Claim 4.6: (i) Suppose, for a contradiction, that G' contains an induced subgraph H that is isomorphic to a claw or a paw. Since G is (claw, paw)-free, H contains the edge ad. By symmetry, we may assume that either $d_H(a) = 3$ or $d_H(a) = d_H(d) = 2$. If $d_H(a) = 3$, then $G[(V(H) \setminus \{d\}) \cup \{b\}]$ is isomorphic to a claw or to a paw, which is a contradiction. If $d_H(a) = d_H(d) = 2$, and x is the common neighbor of a and d in H, then $G[\{a, b, c, x\}]$ is isomorphic to a paw, which is a contradiction.

(ii) First, we assume that M is an allowed decycling matching of (G, F). Since $G[\{a, b, c, d\}] - (M \cap E(G[\{a, b, c, d\}]))$ is a forest, we obtain that $M \cap E(G[\{a, b, c, d\}])$ is either $\{ab, cd\}$ or $\{ac, bd\}$, that is, $ab, cd \notin F$ or $ac, bd \notin F$. Since $G - M$ contains either the path $abcd$ or the path $acbd$, it follows that $M \setminus E(G[\{a, b, c, d\}])$ is an allowed decycling matching of (G', F').

Next, we assume that M' is an allowed decycling matching of (G', F'), and that, by symmetry, $ab, cd \notin F$. Since $ad \in F'$, we obtain that $ad \notin M'$. This implies that $M' \cup \{ab, cd\}$ is an allowed decycling matching of (G, F).

Iteratively applying the reductions captured by the above claims, clearly allows to decide in polynomial time whether (G, F) has an allowed decycling matching, and to determine such a matching in polynomial time if it exists. □

Now we show a polynomial time algorithm for a P_5-free graph G. By a result of Liu et al. [102], G has a dominating induced cycle of length 5 or a dominating clique.

Theorem 4.7 Allowed Decycling Matching *can be solved in polynomial time for P_5-free graphs.*

Proof. Let G be a P_5-free graph and let F be a set of edges of G. Again, we may assume that G is connected.

First, we assume that G has a dominating induced cycle C of length 5. Since G is P_5-free, every vertex in $V(G) \setminus V(C)$ has at least two neighbors in $V(C)$. This implies that $m(G) \geq 5 + 2(n(G) - 5) = 2n(G) - 5$. If $n(G) \geq 9$, then $m(G) > \frac{3n(G)}{2} - 1$, and Lemma 4.1(ii) implies that (G, F) has no allowed decycling matching. Hence, we may assume that $n(G) \leq 8$ in this case, which implies that Allowed Decycling Matching can be solved in constant time.

Next, we assume that G has a dominating clique C of order p. Lemma 4.1(ii), we may assume that $p \leq 3$. If $p = 1$, then G has a universal vertex u. Now, it follows easily that (G, F) has an allowed decycling matching if and only if one of the following two conditions holds:

- $G - u$ is a graph of maximum degree at most 1 that contains no edge from F.
- u has a neighbor v of degree at most 2 such that $uv \notin F$, and $G - \{u, v\}$ is a graph of maximum degree at most 1 that contains no edge from F.

The first condition is equivalent to the existence of an allowed decycling matching that contains no edge incident with u, while the second condition is equivalent to the existence of an allowed decycling matching that contains the edge uv.

If $p = 3$, then G has a dominating triangle $uvwu$. Clearly, every decycling matching must contain one of the three edges of $uvwu$. Furthermore, u, v, and w belong to the same component of $G - M$ for every matching M of G. Considering the polynomially many possibilities to select one of the three edges of $uvwu$ as well as at most one further edge incident to a vertex in C, both forming a matching that does not intersect F, we can deduce a polynomial number of efficiently checkable conditions, very similar to those explicitly stated in the case $p = 1$, such that (G, F) has an allowed decycling matching if and only if one of these conditions is satisfied.

Finally, if $p = 2$, then we may assume that no vertex in $V(G) \setminus C$ has two neighbors in C; since otherwise, G has a dominating triangle. This implies that if uv is a dominating edge of G, then $V(G)$ is partitioned into $\{u, v\}$, $N_G(u) \setminus \{v\}$, and $N_G(v) \setminus \{u\}$. Clearly, every matching contains at most two edges incident with u and v. Again, considering the polynomially many possibilities to select such edges, we can deduce a polynomial number of efficiently checkable conditions such that (G, F) has an allowed decycling matching if and only if one of these conditions is satisfied. For example, (G, F) has an allowed decycling matching that contains uv if and only if $uv \notin F$, and there is a set E containing at most one edge between $N_G(u) \setminus \{v\}$ and $N_G(v) \setminus \{u\}$ such that $(G - \{u, v\}) - E$ has maximum degree at most 1, and contains no edge from F. Clearly, this implies the desired statement. □

Next we prove that there exists a polynomial time algorithm for chordal graphs.

Theorem 4.8 Allowed Decycling Matching *can be solved in polynomial time for chordal graphs.*

Proof. Let G be a chordal graph and let F be a set of edges of G. Again, we may assume that G is connected. Furthermore, we may assume that G does not contain bridges, that is, every block of G has order at least 3. If G contains a cycle C of length ℓ at least 5, then, since G is chordal, $m(G[V(C)]) \geq 2\ell - 3 > \frac{3\ell}{2} - 1$, and Lemma 4.1(ii) implies that G does not have an allowed decycling matching. Hence, every cycle of G has length at most 4.

Let B be a block of G of order at least 4. Since G is chordal, B contains a triangle $abca$. Since B has order at least 4, we may assume that b has a neighbor d distinct from a and c. Since G has no cycle of length at least 5, considering a shortest path in $B - b$ between d and a vertex in $\{a, c\}$ implies that we may assume, by symmetry, that c and d are adjacent. By Lemma 4.1(ii), G does not contains K_4, which implies that $G[\{a, b, c, d\}]$ is $K_4 - e$. Suppose that B has order at least 5. This implies that some vertex x in $V(B) \setminus \{a, b, c, d\}$ has a neighbor y in $\{a, b, c, d\}$. Considering a shortest path in $B - y$ between x and a vertex in $\{a, b, c, d\} \setminus \{y\}$, and using the absence of cycles of length at least 5, we obtain that x is adjacent to a and d. Hence, $m(G[\{a, b, c, d, x\}]) \geq 7 > \frac{3 \cdot 5}{2} - 1$, and Lemma 4.1(ii) implies that G does not have an allowed decycling matching. Hence, we may assume that every block of G is either a triangle or isomorphic to $K_4 - e$.

Clearly, we may assume that G has at least 2 blocks. Let B be an endblock of G, that is, B contains exactly one cutvertex.

First, we assume that B is a triangle $abca$ and that c is the cutvertex of B. If $ab, bc, ca \in F$, then (G, F) does not have an allowed decycling matching. If $ab \notin F$, then (G, F) has an allowed decycling matching if and only if $(G - \{a, b\}, F \setminus$

$\{ab, bc, ca\})$ has an allowed decycling matching. If $ab \in F$ but $ac \not\in F$ or $bc \not\in F$, then (G, F) has an allowed decycling matching if and only if $\big(G - \{a, b\}, (F \setminus \{ab, bc, ca\}) \cup \{cx : x \in N_G(c) \setminus \{a, b\}\}\big)$ has an allowed decycling matching.

Next, we assume that B is isomorphic to $K_4 - e$. Let x denote the cutvertex contained in B. It follows that (G, F) has an allowed decycling matching if and only if one of the two perfect matchings of B does not intersect F, and $\big(G - (V(B) \setminus \{x\}), (F \setminus E(B)) \cup \{xy : y \in N_G(x) \setminus V(B)\}\big)$ has an allowed decycling matching.

Iteratively applying these reductions allows to decide in polynomial time whether (G, F) has an allowed decycling matching, and also, to determine such a matching in polynomial time if it exits. □

Next we prove that there exists a polynomial time algorithm for C_4-free distance hereditary graphs. By results of Bandelt and Mulder [11], this implies that G has an endvertex, or that G contains two vertices a and b with either $N_G[a] = N_G[b]$ or $N_G(a) = N_G(b)$.

Theorem 4.9 Allowed Decycling Matching *can be solved in polynomial time for C_4-free distance hereditary graphs.*

Proof. Let G be a (connected) C_4-free distance hereditary graph and let F be a set of edges of G. Clearly, we may assume that G has order at least 3.

Clearly, if G has an endvertex x, then (G, F) has an allowed decycling matching if and only if $(G - x, F \cap E(G - x))$ has an allowed decycling matching. Hence, we may assume that G has no endvertex, and that the second case occurs. If $|N_G(a) \setminus \{b\}| \geq 3$, then, since G is C_4-free, we obtain $m(G[N_G[a]]) > \frac{3}{2}|N_G[a]| - 1$, and Lemma 4.1(ii) implies that (G, F) does not have an allowed decycling matching. If $|N_G(a) \setminus \{b\}| = 2$, then, since G is C_4-free, $G[N_G[a]]$ is isomorphic either to K_4 or to $K_4 - e$. In the first case, Lemma 4.1(ii) implies that (G, F) does not have an allowed decycling matching, and, in the second case, (G, F) has an allowed decycling matching, if and only if some perfect matching of $G[N_G[a]]$ does not intersect F, and $(G - \{a, b\}, (F \cap E(G - \{a, b\})) \cup \{uv \in E(G - \{a, b\}) : u \in N_G[a] \setminus \{a, b\}\})$ has an allowed decycling matching. If $|N_G(a) \setminus \{b\}| = 1$, then, since G does not have an endvertex, a and b are adjacent and lie on a triangle with the unique vertex c in $N_G(a) \cap N_G(b)$. If $ab, bc, ca \in F$, then (G, F) does not have an allowed decycling matching, if $ab \not\in F$, then (G, F) has an allowed decycling matching if and only if $(G - \{a, b\}, F \cap E(G - \{a, b\}))$ has an allowed decycling matching, and if $ab \in F$ and $ac \not\in F$, then (G, F) has an allowed decycling matching if and only if $(G - \{a, b\}, (F \cap E(G - \{a, b\})) \cup \{uc \in E(G) : u \in N_G(c) \setminus \{a, b\}\})$ has an allowed decycling matching. Iteratively applying these reductions allows to decide in polynomial time whether (G, F) has an allowed decycling matching, and also, to determine such a matching in polynomial time if it exits. □

4.2 Obtaining a Bipartite Graph

In this section we deal with the complexity of the following decision problem.

ODD DECYCLING MATCHING
Input: A finite, simple, and undirected graph G.
Question: Does $G \in \mathcal{BM}$?

A more restricted version of this problem is considered by Schaefer [129]. He deals with the problem of determining whether a given graph G admits a 2-coloring of the vertices so that each vertex has *exactly* one neighbor with same color as itself. We can see that the removal of the set of edges whose endvertices have same color, which is a perfect matching of G, generates a bipartite graph. Schaefer proved that such a problem is NP-complete even for planar cubic graphs.

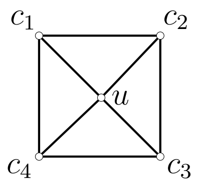
With respect to the minimization version, the edge-deletion decision problem in order to obtain a bipartite graph is analogous to SIMPLE MAX CUT, which was proved to be NP-complete by Garey, Johnson and Stockmeyer [76]. Yannakakis [139] proved its NP-completeness even for cubic graphs.

ODD DECYCLING MATCHING can also be seen as another problem. A graph G is $(d_1, \ldots, d_k)$-*colorable* if $V(G)$ can be partitioned into $V_1, \ldots, V_k$, such that the induced subgraph $G[V_i]$ has maximum degree at most d_i, for all $1 \leq i \leq k$. This is a generalization of the classical *proper k-coloring*, when every $d_i = 0$, and the *d-improper k-coloring*, when every $d_i = d \geq 1$. It is clear to see that $G \in \mathcal{BM}$ if and only if G is $(1,1)$-colorable. Lovász [103] proved that if a graph G satisfies $(d_1+1)+(d_2+1)+\cdots+(d_k+1) \geq \Delta(G)+1$ then G is $(d_1, \ldots, d_k)$-colorable, where $\Delta(G)$ denotes the *maximum degree* of G. This result shows that every subcubic graph is $(1,1)$-colorable and thus belongs to $\mathcal{BM}$. Borodin, Kostochka, and Yancey [24] studied the $(1,1)$-colorable graphs with respect to the sparseness parameter $mad(G) = \max\left\{\frac{2|E(H)|}{|V(H)|}, \text{ for all } H \subseteq G\right\}$. They proved that every graph G with $mad(G) \leq \frac{14}{5}$ is $(1,1)$-colorable, where this bound is sharp. Moreover, they defined the parameter $\rho(G) = \min_{S \subseteq V(G)} \rho_G(S)$, such that $\rho_G(S) = 7|S| - 5|E(G[S])|$. They showed that G is $(1,1)$-colorable if $\rho(G) \geq 0$. Finally, they also proved that every planar graph with *girth* (the size of the smallest cycle of G) at least 7 is $(1,1)$-colorable. This is the best result concerning $(1,1)$-coloring of planar graphs.

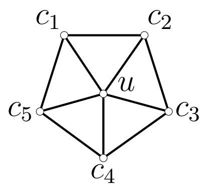
For a matching $M \subseteq E(G)$, we say that M is an *odd decycling matching* of G if $G - M$ is bipartite. Let $\mathcal{BM}$ denote the set of all graphs admitting an odd decycling matching.

4.2.1 Preliminaries

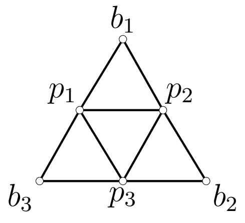
A *diamond* is the graph obtained by removing one edge from the K_4. Let W_k be the *wheel graph* of order k, that is, the graph containing a vertex v, called *central*,



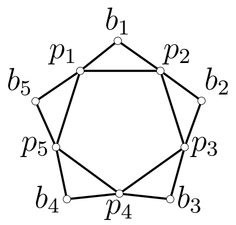
(a) The W_4.



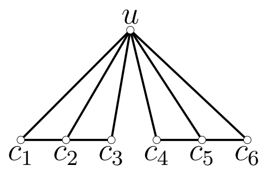
(b) The W_5.



(c) The 3-pool.



(d) The 5-pool



(e) Two diamonds sharing a vertex of degree 3.

Figure 4.1: Some examples of forbidden subgraphs.

and a cycle C of order k, such that v is adjacent to all vertices of C.

We say that a graph is a *k-pool* if it is formed by k triangles edge disjoint whose bases induce a C_k. Formally, a k-pool is obtained from a cycle $C = \{v_1, v_2, \dots, v_{2k}\}$ ($k \geq 3$), such that the odd-indexed vertices induce a cycle $p_1p_2 \dots p_kp_1$, called *internal cycle* of the k-pool, where $p_i = v_{2i-1}$, $1 \leq i \leq k$. The even-indexed vertex b_i is the i-th-*border* of the k-pool, where $\{b_i\} = N_C(p_i) \cap N_C(p_{i+1})$ and $i+1$ is taken modulo $2k$. Fig. 4.1c and Fig. 4.1d represent the 3-pool and 5-pool, respectively.

Clearly, every graph $G \in \mathcal{BM}$ admits a proper 4-coloring. Hence every graph in $\mathcal{BM}$ is K_5-free. More precisely, every graph in $\mathcal{BM}$ is W_4-free, which is depicted in Fig 4.1a. Hence some proper 4-colorable graphs do not admits an odd decycling matching. Fig. 4.1 shows some others examples of forbidden subgraphs. Lemma 4.10 collects some properties of graphs in $\mathcal{BM}$.

Lemma 4.10 *Given a graph G in $\mathcal{BM}$ and an odd decycling matching M of G, the following assertions are true.*

(i) *If G has a diamond D as a subgraph, then M contains no edge $e \notin E(D)$ incident to only one vertex of degree three of D.*

(ii) *$G[N_G(v)]$ cannot contain two disjoint P_3, for every $v \in V(G)$.*

(iii) *G cannot contain a W_k as a subgraph, for all $k \geq 4$.*

(iv) *G cannot contain a k-pool as a subgraph, for all odd $k \geq 3$.*

Proof. (i) Let $G \in \mathcal{BM}$ be a graph that contains a diamond D as a subgraph, such that $V(D) = \{u, v_1, v_2, v_3\}$ and $d_D(u) = d_D(v_2) = 3$. We can see that $M \cap E(D)$

equals to exactly one of the following sets: $\{uv_1, v_2v_3\}$, $\{v_1v_2, uv_3\}$, $\{uv_2\}$. For each of such sets, both u and v_2 are matched by M. Hence M cannot contain any edge $e \notin E(G[V(D)])$ incident to only u or v_2.

(ii) Let $v \in V(G)$ such that $G[N_G(v)]$ contains two disjoint P_3, P and P'. It follows that $G[\{v\} \cup P]$ and $G[\{v\} \cup P']$ are diamonds that share a vertex of degree at least three. By (i) the statement holds.

(iii) Suppose for a contradiction that G contains a subgraph H isomorphic to a wheel graph W_k, $k \geq 4$. Let $V(H) = \{u, v_1, v_2, \ldots, v_{k-1}, v_k\}$, such that u is adjacent to all vertices of the cycle $C = v_1v_2 \ldots v_kv_1$. If $k \geq 6$, then u contains two disjoint P_3 in its neighborhood, and thus it follows by (ii) that $k \leq 5$. In this case, it can be easily verified that W_4 and W_5 are forbidden subgraphs.

(iv) Suppose, for a contradiction, that G contains a subgraph H isomorphic to a k-pool, for some odd $k \geq 3$. Let $C = \{p_1p_2 \ldots p_kp_1\}$ be its internal cycle and let $B = \{b_1, b_2, \ldots, b_k\}$ be the vertices of the border of H, such that $\{p_ib_i, p_{i+1}b_i\} \subset E(H)$, for all $1 \leq i \leq k$ modulo k. Clearly M must contain some edge of C and one edge of every triangle $p_ib_ip_{i+1}$. W.l.o.g., consider $p_1p_2 \in M \cap E(C)$. This implies that M contains no edge in $\{p_2b_2, p_2p_3, p_1b_k, p_1p_k\}$. Therefore, p_kb_k and p_3b_2 must be in M, which forbids two more edges from the triangles $p_{k-1}b_{k-1}p_k$ and $p_3b_3p_4$. Continuing this process, it follows that $c_{\lfloor \frac{k+3}{2} \rfloor}$, which is at the same distance of p_1 and p_2 in C, must contain two incident edges in M, a contradiction. □

4.2.2 A linear Time Algorithm for Subcubic Graphs

Bondy and Locke [23] presented the following lemma, which was also obtained by Erdős [67] by induction on $n(G)$.

Lemma 4.11 *(Bondy and Locke [23]) Let G be a graph and let B be a largest bipartite subgraph of G. Then $d_B(v) \geq \frac{1}{2}d_G(v)$, for every $v \in V(G)$.*

Lemma 4.11 shows that every subcubic graph G admits an odd decycling matching, since every vertex has at most one incident edge not in a largest bipartite subgraph of G. This result was also obtained by Lovász [103] with respect to 1-improper 2-coloring of graphs with maximum degree at most 3.

Consider a bipartition of $V(G)$ into sets A and B. For every vertex v, we say that v is of type (a, b) if $d_{V(G)\setminus X}(v) = a$ and $d_X(v) = b$, where X is the part (either A or B) which contains v. We present a linear algorithm to find an odd decycling matching of subcubic graphs, Algorithm 4.

Theorem 4.12 *Algorithm 4 returns in linear time an odd decycling matching for subcubic graphs.*

Algorithm 4: A linear time algorithm that determines an odd decycling matching for subcubic graphs.

```
Data: A subcubic graph $G$.
Result: An odd decycling matching $M$ of $G$.
$A \leftarrow$ A maximal independent set of $G$;
$B \leftarrow V(G) \setminus A$;
$M \leftarrow \emptyset$;
while exists a vertex $v \in B$ of type $(1,2)$, with respect to $A$ and $B$, do
    $u \leftarrow N_{G[A]}(v)$;
    if $u$ is of type $(3,0)$ then
        $B \leftarrow B \setminus \{v\}$;
        $A \leftarrow A \cup \{v\}$;
    else
        $B \leftarrow \{B \cup \{u\}\} \setminus \{v\}$;
        $A \leftarrow \{A \cup \{v\}\} \setminus \{u\}$;
$M \leftarrow$ all edges of $G[A] \cup G[B]$;
return $M$;
```

Proof. Let A be a maximal independent set of G. Let $B = V(G) \setminus A$. In this case, every vertex of A is of type $(k, 0)$ and there is no vertex in B of type $(0, k)$, $k \in \{1, 2, 3\}$. Therefore, if there exists a vertex v of type (a, b) with $a < b$, then it must be in B and be of type $(1, 2)$. In order to prove the correctness of Algorithm 4, it is sufficient to show that the operations on lines 7–8 and 10–11 do not generate vertices of type (a, b) with $a < b$.

Let $\{u\} = N_{G[A]}(v)$. If u is of type $(3, 0)$, then v is moved from B to A by lines 7–8. In this case, it follows that both u and v are vertices of type $(2, 1)$ after the line 8. If u is not of type $(3, 0)$, then the lines 10–11 modify the types of u and v as follows.

- If u is of type $(1, 1)$, then u and v are modified to type $(2, 0)$ and $(3, 0)$, respectively;
- If u is of type $(1, 0)$, then u and v are modified to type $(1, 0)$ and $(3, 0)$, respectively;
- If u is of type $(2, 0)$, then u and v are modified to type $(1, 1)$ and $(3, 0)$, respectively;
- If u is of type $(2, 1)$, then u and v are modified to type $(2, 1)$ and $(3, 0)$, respectively.

We can see that each neighbor w of u in the same part $X \in \{A, B\}$ of u loses exactly one neighbor (that is u) in $G[X]$. Moreover, w receives at most one new neighbor (that is v) in $G[X]$. The same occurs for every neighbor of v in $V(G) \setminus X$. Therefore, in any case it is not obtained vertices of type (a, b) with $a < b$, which implies that the Algorithm 4 finishes. □

Despite the simplicity of Algorithm 4, determining the size of a minimum odd decycling matching of subcubic graphs is NP-hard, since this problem becomes analogous to MAX CUT [75] for such a class.

4.2.3 NP-Completeness for Odd Decycling Matching

In this section we prove that Odd Decycling Matching is NP-complete even for planar graphs of maximum degree at most 4. We organize the proof in three parts. In the first one we show some polynomial time reductions from Not-All-Equal 3-SAT (NAE-3SAT) [129] and Positive Planar 1-In-3-SAT [110]. In the second part we prove that Odd Decycling Matching is NP-complete for graphs with maximum degree at most 4. This proof is a more intuitive and easier to understand the gadgets and construction of the next part. The third part presents a proof that Odd Decycling Matching is NP-complete even for planar graphs with maximum degree at most 5. Finally, the proof finishes as a corollary from the previous results by just slightly modifying the used gadgets.

Preliminaries

Let F be a Boolean formula in CNF with set of variables $X = \{x_1, x_2, \ldots, x_n\}$ and set of clauses $C = \{c_1, c_2, \ldots, c_m\}$. The *associated graph* of F, $G_F = (V, E)$, is the bipartite graph such that there exists a vertex for every variable and clause of F, where (X, C) is a bipartition of $V(G_F)$ into independent sets. Furthermore, there exists an edge $x_i c_j \in E(G_F)$ if and only if c_j contains either x_i or $\overline{x_i}$. We say that F is *planar* if its associated graph is planar. In order to obtain a polynomial reduction, we consider the following decision problems, which are NP-complete.

Not-All-Equal 3-SAT (NAE-3SAT) [129]
Input: A Boolean formula in 3-CNF, F.
Question: Is there a truth assignment to the variables of F, in which each clause has one literal assigned true and one literal assigned false?

Positive Planar 1-In-3-SAT [110]
Input: A planar Boolean formula in 3-CNF, F, with no negated literals.
Question: Is there a truth assignment to the variables of F, in which each clause has exactly one literal assigned true?

In order to prove the NP-completeness of Odd Decycling Matching, we first present a polynomial time reduction from NAE-3SAT and Positive Planar 1-In-3-SAT to the following decision problems, respectively:

NAE-3SAT$_3$
Input: A Boolean formula in CNF, F, where each clause has either 2 or 3 literals, each variable occurs at most 3 times, and each literal occurs at most twice.
Question: Is there a truth assignment to the variables of F in which each clause has at least one literal assigned true and at least one literal assigned false?

Planar 1-In-3-SAT$_3$
Input: A planar Boolean formula in CNF, F, where each clause has either 2 or 3 literals and each variable occurs at most 3 times. Moreover, each positive literal occurs at most twice, while every negative literal occurs at most once in F.
Question: Is there a truth assignment to the variables of F in which each clause has exactly one true literal?

Theorem 4.13

- NAE-3SAT$_3$ *is* NP*-complete.*
- Planar 1-In-3-SAT$_3$ *is* NP*-complete.*

Proof. Since verifying whether a graph is planar can be done in linear time [89], as well as whether a formula in 3-CNF has a truth assignment, both problems are in NP.

Let F be a Boolean formula in 3-CNF such that $X = \{x_1, x_2, \ldots, x_n\}$ denotes the set of variables and $C = \{c_1, c_2, \ldots, c_m\}$ is the set of clauses of F. We construct a formula F' from F as follows. For a vertex $x_i \in V(G_F[X])$, let $d_{G_F}(x_i)$ be the degree of x_i in G_F. For such a variable x_i with $d_{G_F}(x_i) = k \geq 3$, we create k new clauses c_i^j of size 2, and k new variables x_i^z as follows:

$$c_i^j = \begin{cases} \left(x_i^j,\ \overline{x_i^{j+1}}\right) & \text{, if } j \in \{1, \ldots, k-1\}; \\ \left(x_i^k,\ \overline{x_i^1}\right) & \text{, if } j = k. \end{cases}$$

In addition, we replace the j^{th} $(1 \leq j \leq k)$ occurrence of the variable $x_i \in X$ by an occurrence of a variable x_i^j, where a literal x_i (resp. $\overline{x_i}$) is replaced by a literal x_i^j $\left(\text{resp. } \overline{x_i^j}\right)$.

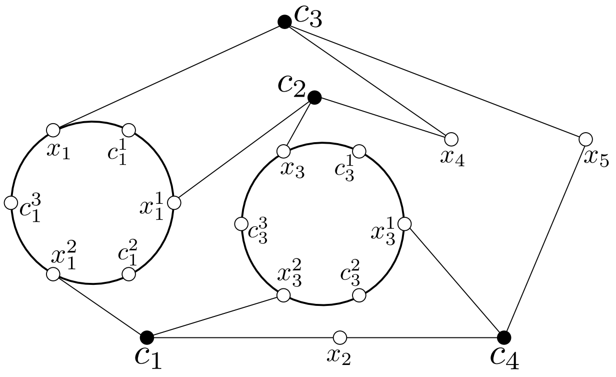


Figure 4.2: The associated graph $G_{F'}$ obtained from $F = (x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_3 \vee x_4) \wedge (x_1 \vee x_4 \vee x_5) \wedge (x_2 \vee x_3 \vee x_5)$. The black vertices correspond to clauses.

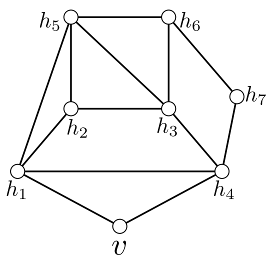
Let S be the set of all vertices $x_i \in V(G_F[X])$ with $d_{G_F}(x_i) = k \geq 3$. For such a vertex $x_i \in S$, let $X_i = \{x_i^1, \ldots, x_i^k\}$ and $C_i = \{c_i^1, \ldots, c_i^k\}$.

Note that, the associated graph $G_{F'}$ can be obtained from G_F by replacing the corresponding vertex of $x_i \in S$ by a cycle of length $2d_{G_F}(x_i)$ induced by the corresponding vertices of the new clauses in C_i and the new variables in X_i. In addition, for each $x_i \in S$ and $c_j \in N_{G_F}(x_i)$ an edge $x_i^t c_j$ is added in $E(G_{F'})$, such that every corresponding vertex $x_i^t \in X_i$ has exactly one neighbor $c_j \notin C_i$. Figure 4.2 shows an example of the transformation for a Boolean formula.

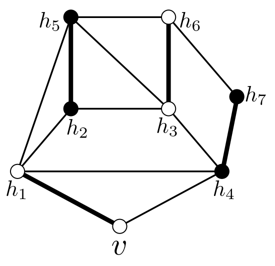
As we can see, every variable x occurs at most 3 times in the clauses of F', since every variable x_i with $d_{G_F}(x_i) \geq 3$ is replaced by $d_{G_F}(x_i)$ new variables that are in exactly 3 clauses of F'. By the construction, each literal occurs at most twice. Moreover, if F has no negative literals, then only the new variables have a negated literal and each one occurs exactly once in F'.

Now, it remains to show that if G_F is planar then we can construct F' as a planar formula. Consider a planar embedding Ψ of G_F, we construct $G_{F'}$ replacing each corresponding vertex $x_i \in S$ by a cycle of length $2d_{G_F}(x_i)$, as described above. After that, in order to preserve the planarity, we can follow the planar embedding Ψ to add a matching between vertices corresponding to variables in such a cycle and vertices corresponding to clauses $c_j \notin C_i$ and that $x_i \in c_j$. Such a matching indicates in which clause of C_i a given new variable will replace x_i in F'. Thus, without loss of generality, if G_F is planar then we can assume that F' is planar as well.

Let F be an instance of NAE-3SAT (resp. Positive Planar 1-In-3-SAT) such that $X = \{x_1, \ldots, x_n\}$ denotes its set of variables and $C = \{c_1, c_2, \ldots, c_m\}$ its set of clauses. Let F' be the formula obtained from F by the above construction. As we can observe, for any truth assignment of F', all $x_i^t \in X_i$ (for a given variable x_i of F) have the same value. Therefore, any clause of F' containing exactly two literals has true and false values. At this point, it is easy to see that F has a not-all-equal (resp. 1-in-3) truth assignment if and only if F' has a not-all-equal (resp. 1-in-3)



(a) The head graph H.



(b) The odd decycling matching of H.

Figure 4.3: The head graph and its odd decycling matching M, which is represented by the stressed edges. Vertices with same color belong to the same part in the bipartition of $G - M$.

truth assignment. □

Now we show the NP-completeness of Odd Decycling Matching by a reduction from NAE-3SAT$_3$. The next simple lemma is used in the correctness of our reduction.

Let us call the graph depicted in Figure 4.3a by *head*. Vertex v is the *neck* of the head. Given a graph G, the next lemma shows that such a structure is very useful to ensure that some edges cannot be in any odd decycling matching of G.

Lemma 4.14 *Let G be a graph that contains an induced subgraph H isomorphic to a head graph, whose neck is v. Then all edges not in H incident to v cannot be in any odd decycling matching of G. Moreover H admits only one odd decycling matching.*

Proof. Let M be an odd decycling matching of G. Suppose for a contradiction that there exists an edge e incident to v, such that e contains an endvertex not in H. In this case, we get that vh_1 and vh_4 does not belong to M, which implies that $h_1h_4 \in M$. By the triangle $h_1h_2h_5$, it follows that h_2h_5 must be in M. Hence the cycle $vh_1h_2h_3h_4v$ remains in $G - M$, a contradiction.

Now suppose that $vh_4 \in M$. In this case, the edge h_1h_2 cannot be in M, otherwise the cycle $h_1h_4h_3h_2h_5h_1$ survives in $G - M$. In the same way, the edge $h_1h_5 \notin M$, otherwise the cycle $h_1h_2h_5h_3h_4h_1$ is not destroyed by M. Therefore we get that h_2h_5 must be in M, which implies that $h_3h_6 \in M$. Hence the cycle $h_5h_3h_4h_7h_6h_5$ belongs to $G - M$. Since the triangle $h_1h_2h_5$ has no edge in M, it is not destroyed by M, a contradiction.

Finally, we get that vh_1 must be in M, which implies that $h_2h_5 \in M$ as well. Therefore, it follows that h_3h_6 must be in M. Hence h_4h_7 also must be in M, which turns the graph bipartite. Since all choices of the edges of M are necessary, we get that there is only one possible odd decycling matching of H, which is perfect. Fig. 4.3b shows such a matching. This concludes the proof. □

NP-Completeness for Graphs with Maximum Degree at Most 4

With Lemma 4.14 we can establish the NP-completeness of Odd Decycling Matching. Remember that graphs in $\mathcal{BM}$ are all 4-colorable. The next result shows that the NP-completeness is also obtained even for 3-colorable graphs and bounded degree graphs. The circles with an H in the figures represent an induced subgraph isomorphic to the head graph, whose neck is the vertex touching the circle. By simplicity, this pattern will be used in the remaining figures whenever possible.

Theorem 4.15 Odd Decycling Matching *is NP-complete even for 3-colorable graphs with maximum degree at most 4.*

Proof. We prove that Odd Decycling Matching is NP-complete by a reduction from NAE-3SAT$_3$ Let F be an instance of NAE-3SAT$_3$, with $X = \{x_1, x_2, \ldots, x_n\}$ and $C = \{c_1, c_2, \ldots, c_m\}$ be the sets of variables and clauses of F, respectively. We construct a graph $G = (V, E)$ as follows:

- For each variable $x_i \in X$, we construct a variable gadget G_{x_i}. Such a gadget consists on a diamond D with a head, whose neck is the vertex u_i of degree two in D. The vertices of degree three in D, d_i^1 and d_i^2, represent the literal x_i, while the last one, d_i^3, of degree two represents the negative literal $\overline{x_i}$. Figure 4.4 shows the variable gadget G_{x_i}.

- For each clause $c_j \in C$, we associate a clause gadget G_{c_j}. If c_j contains three literals, then G_{c_j} is a triangle with vertices c_j^1, c_j^2, and c_j^3. Moreover, each vertex c_j^k is adjacent to a *linking vertex* ℓ_j^k, $k \in \{1, 2, 3\}$, which is a neck of a head H. Such clause gadget is showed in Figure 4.5b. In a similar way, if c_j has size two, then G_{c_j} is as depicted in Figure 4.5a, where ℓ_j^1 and ℓ_j^2 are the vertices that connect G_{c_j} to the gadgets of the variables contained in c_j.

- We link a clause gadget G_{c_j} to a variable gadget G_{x_i}, such that $x_i \in c_j$, as follows. If c_j contains the positive literal x_i, then add one edge between a linking vertex ℓ_j^k to either d_i^1 or d_i^2, otherwise we add the edge $\ell_j^k d_i^3$, for some $1 \leq k \leq 3$.

Since the Head graph is 3-colorable, clearly the above construction generates also a 3-colorable graph. Next we prove that F has a truth assignment if and only if the graph G obtained form the above construction has an odd decycling matching. If F has a truth assignment ϕ, then each clause c_j contains at least one true literal and at least one false literal. For such a clause, we associate true to c_j^k if and only if its corresponding literal is true in ϕ. In the same way, for every variable gadget G_{x_i}, we associate true to d_i^1 and d_i^2 if and only if the positive literal x_i is true in ϕ. Therefore,

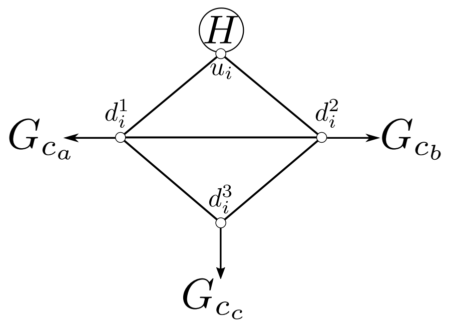


Figure 4.4: Variable gadget G_{c_j} in Theorem 4.15.

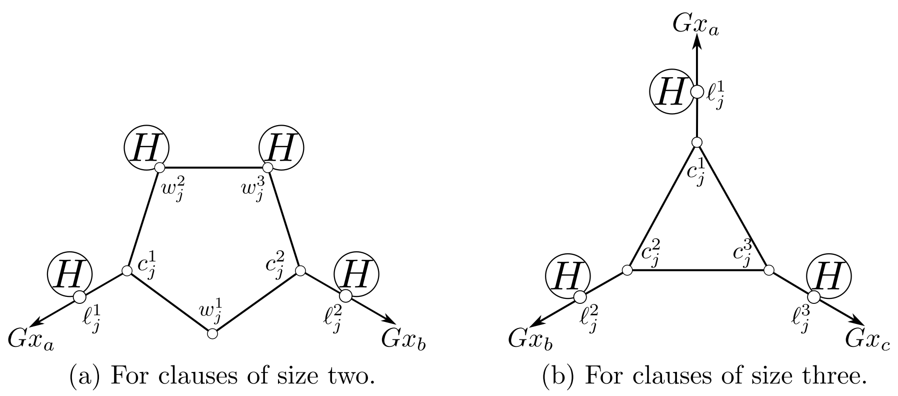


Figure 4.5: Clause gadget G_{c_j} in Theorem 4.15.

we can construct a bipartition of $V(G)$ into sets T and F, that represent the literal assigned true and false, respectively, as follows.

- For each clause gadget G_{c_j} of 3 literals, remove the edge $c_j^z c_j^w$ if $\phi(c_j^z) = \phi(c_j^w)$, $1 \leq z \neq w \leq 3$;
- For each clause gadget G_{c_j} of 2 literals, remove either the edge $w_j^1 c_j^1$ or $w_j^1 c_j^2$;
- For every variable gadget G_{x_i}, remove the edge $d_i^1 d_i^2$;
- For each induced head H, remove edges as in Fig. 4.3b.

It is not hard to see that the obtained graph is bipartite, since each linking vertex ℓ_j^k is in the opposite set of c_j^k and of d_i^z, such that $\ell_j^k d_i^z \in E(G)$. Moreover, c_j^1 and c_j^2 are in opposite sets, for every clause of length 2. Since the removed edges are clearly a matching in G, it follows that $G \in \mathcal{BM}$.

Now we consider that $G \in \mathcal{BM}$. By Lemma 4.14, it follows that $d_i^1 d_i^2$ must be in any odd decycling matching of G, for every variable gadget G_{x_i}. Analogously, either $w_j^1 c_j^1$ or $w_j^1 c_j^2$ and exactly one edge $c_j^k c_j^z$ must be included in any odd decycling matching of G, for every clause gadget G_{c_j} of 2 and 3 literals, respectively. Therefore, for an odd decycling matching of G, we can associate to the parts of the bipartition of $G - M$ as true and false. Thus, it follows that:

- d_i^1 and d_i^2 are in the same part, while d_i^3 is in the opposite one, for every variable gadget G_{x_i};
- c_j^1 and c_j^2 are in different parts, for every clause gadget G_{c_j} of length 2;
- All the vertices c_j^k are not in the same part, for every clause gadget G_{c_j} of length 3;

Hence, every clause has at least one true and one false literal, which implies that F is satisfiable. $\square$

Since NAE-3SAT is polynomial time solvable for planar graphs [108], the previous construction cannot be planar. Moreover planar graphs are classical 4-colorable graphs. Hence it is interesting to know what happens in such a class. The next Subsection deals with this problem.

NP-Completeness for Planar Graphs

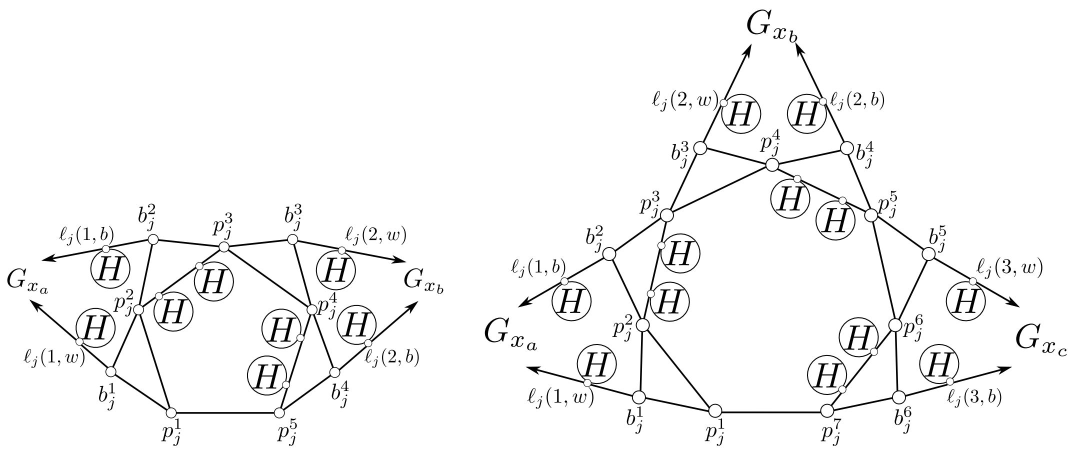
Now we will show that Odd Decycling Matching remains NP-complete even for planar graphs. We prove the NP-completeness by a reduction from Planar 1-In-3-SAT$_3$. In order to prove this result, next we give a useful lemma.

Lemma 4.16 *Let b be a border of an odd k-pool graph G, such that c_1 and c_k are its neighbors in G. It follows that every odd decycling matching of $G - b$ must contain exactly one edge of the internal cycle, which is different from c_1c_k. Moreover, there is only one odd decycling matching for each such an edge.*

Proof. Let $C = p_1p_2 \dots p_kp_1$ be the internal cycle of G and let b_i be the i-th-border of G, such that $N_G(b_i) = \{p_i, p_{i+1}\}$, $1 \leq i \leq k-1$. Since C has odd length, it follows that every odd decycling matching of G contains at least one edge of C.

Suppose for a contradiction that G has an odd decycling matching M containing p_1p_k. In this case, we get that the edges in $\{p_1p_2, p_1b_1, p_kp_{k-1}, p_kb_{k-1}\}$ cannot be in M. Therefore M must contain the edges b_1p_2 and $b_{k-1}p_{k-1}$. In the same way, we can see that the edges $\{p_2p_3, p_2b_2, p_{k-1}p_{k-2}, p_{k-1}b_{k-2}\}$ are not in M. Hence, it can be seen that all edges indent to $p_{\frac{k-1}{2}}$ are forbidden to be in M, which implies that the triangles $p_{\frac{k-2}{2}}p_{\frac{k-1}{2}}b_{\frac{k-2}{2}}$ and $p_{\frac{k-1}{2}}p_{\frac{k+1}{2}}b_{\frac{k-1}{2}}$ have no edge in M, a contradiction by the choice of M.

Let p_ip_{i+1} be an edge of C contained in an odd decycling matching M of G. In a same fashion, the edges in $\{p_ip_{i-1}, p_ib_{i-1}, p_ip_{i+1}, p_ib_{i+1}\}$ cannot be in M. Following this pattern, we can see that every edge p_jb_j must be in M, for every $1 \leq j \leq i-1$. Furthermore, it follows that $b_zp_{z+1} \in M$, for every $i+1 \leq z \leq k-1$. Since M contains one edge of every triangle of G, it follows that M is unique, for every edge p_ip_{i+1}. Finally, such an odd decycling matching contains only one edge of C. $\square$



(a) For clauses of size two. (b) For clauses of size three.

Figure 4.6: Clause gadget G_{c_j} in Theorem 4.17.

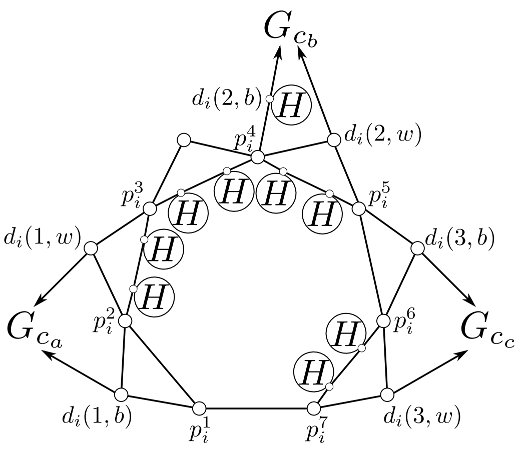


Figure 4.7: Variable gadget G_{x_i} in Theorem 4.17. Each pair of edges with one no endvertex connects G_{x_i} to one clause gadget G_{c_a}, G_{c_b}, or G_{c_c}, where $x_i \in (c_a \cap c_b \cap c_c)$.

Theorem 4.17 Odd Decycling Matching *is NP-complete even for* 3*-colorable planar graphs with maximum degree at most* 5.

Proof. Let F be an instance of Planar 1-In-3-SAT$_3$, with $X = \{x_1, x_2, \ldots, x_n\}$ and $C = \{c_1, c_2, \ldots, c_m\}$ be the sets of variables and clauses of F, respectively. We construct a planar graph $G = (V, E)$ of maximum degree 5 as follows:

- For each clause $c_j \in C$, we construct a gadget G_{c_j} as depicted in Fig. 4.6. Such gadgets are just a 5-pool and a 7-pool less a border for clauses of size 2 and 3, respectively. Moreover, for the alternate edges of the internal cycle we subdivide them twice and append a head graph to each such a new vertex. Finally, we add two vertices $\ell_j(k,w)$ and $\ell_j(k,b)$, such that $b_j^{2k-1}\ell_j(k,w) \in E(G)$ and $b_j^{2k}\ell_j(k,b) \in E(G)$, for $k \in \{1,2,3\}$. For such new vertices, we append a head graph to each one.
- For each variable $x_i \in X$, we construct a gadget G_{x_i} as depicted in Fig. 4.7.

Such a gadget is a 7-pool less a border, where we subdivide the edges $p_i^2p_i^3$, $p_i^3p_i^4$, $p_i^4p_i^5$, and $p_i^6p_i^7$ twice, where every such a new vertex has a pendant head. We rename each border vertex b_i^{2k-1} ($k \in \{1,3\}$) as $d_i(k,b)$ and b_i^{2k} as $d_i(k,w)$, for $k \in \{1,2,3\}$. Moreover we add a new vertex $d_i(2,b)$ adjacent to p_i^4, which has a pendant head graph.

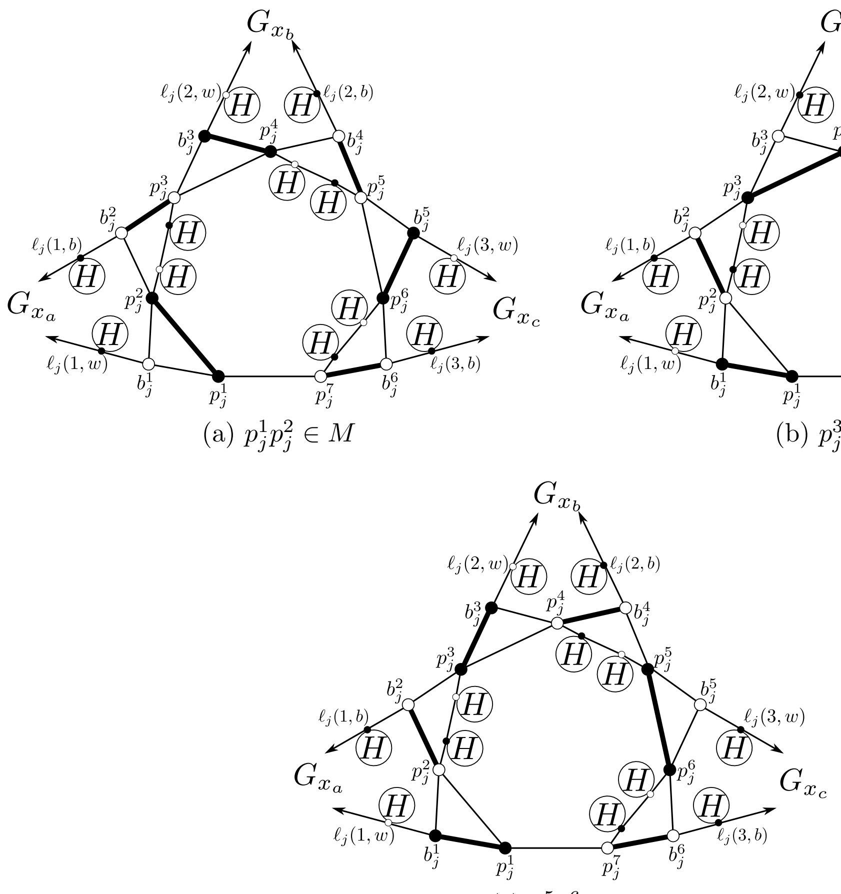
- The connection between clause and variable gadgets are as in Fig. 4.6 and Fig. 4.7, where each pair of arrow head edges in a variable gadget G_{x_i} corresponds to a pair of such edges in a clause gadget G_{c_j}, such that $x_i \in c_j$. More precisely, if $x_i \in c_j$, then we add the edges $\ell_j(k,b)d_i(k',b)$ and $\ell_j(k,w)d_i(k',w)$, for some $k \in \{1,2,3\}$ and for some $k' \in \{1,2\}$. On the other hand, if $\overline{x_i} \in c_j$, then we add the edges $\ell_j(k,b)d_i(3,b)$ and $\ell_j(k,w)d_i(3,w)$, for some $k \in \{1,2,3\}$.

- If a variable occurs only twice in F, then just consider those connections corresponding to the literals of x_i in the clauses of F, such that $d_i(3,b)$ and $d_i(3,w)$ represent $\overline{x_i}$.

Let G be the graph obtained from F by the above construction. We can see that G has maximum degree 5, where the only vertices with degree 5 are those p_i^4, for each variable gadget G_{x_i}. Furthermore, it is clear that G is 3-colorable.

It remains to show that if F is planar (that is, if G_F is planar), then G is planar. Consider a planar embedding ψ of G_F. We replace each variable vertex v_{x_i} of G_F by a variable gadget G_{x_i}, as well as every clause vertex v_{c_j} of G_{c_j} by a clause gadget G_{c_j}. The clause gadgets correspond to clauses of length two or three, which depends on the degree of v_{c_j} in G_F. Since the clause and variable gadgets are planar, we just need to show that the connections among them keep the planarity. Given an edge $v_{x_i}v_{c_j} \in E(G_F)$, we connect G_{x_i} and G_{c_j} by duplicating such an edge as parallel edges $\ell_j(k,w)\ell_i(k',w)$ and $\ell_j(k,b)\ell_i(k',b)$, for some $k \in \{1,2,3\}$ and for some $k' \in \{1,2\}$ or $\ell_j(k,b)\ell_i(3,b)$ and $\ell_j(k,w)\ell_i(3,w)$, for some $k \in \{1,2,3\}$, as previously discussed.

In order to prove that F is satisfiable if and only if $G \in \mathcal{BM}$, we discuss some considerations related to odd decycling matchings of the clause and variable gadgets. By Lemma 4.16, we know that an odd k-pool graph less a border admits one odd decycling matching for each edge of the internal cycle, except that whose both endvertices are adjacent to the removed border. Furthermore, by Lemma 4.14 it follows that each external edge incident to the neck of an induced head cannot be in any odd decycling matching. In this way, Fig. 4.8 shows the possible odd decycling matchings M, given by the stressed edges for the clause gadget G_{c_j} of clauses of length three. The black and white vertex assignment represents the bipartition of $G_{c_j}-M$. Notice that exactly one pair of vertices $\ell_j(k,w)$ and $\ell_j(k,b)$ ($k \in \{1,2,3\}$)



(a) $p_j^1 p_j^2 \in M$ (b) $p_j^3 p_j^4 \in M$ (c) $p_j^5 p_j^6 \in M$

Figure 4.8: All possible configurations given by the removal of an odd decycling matching M, represented by the stressed edges, from a clause gadget G_{c_j} of three literals. The colors in the vertices represent the bipartition of $G_{c_j} - M$.

is such that they have the same color, while the other such pairs have opposite colors. More precisely, we can see that $\ell_j(k, w)$ has the same color for each pair with opposite color vertices as well as $\ell_j(k, b)$, for each odd decycling matching of G_{c_j}. In this way, we can associate one literal x_j^1, x_j^2, and x_j^3 to each pair of vertices $\ell_j(k, w)$ and $\ell_j(k, b)$, $k \in \{1, 2, 3\}$. A similar analysis can be done for clause gadgets of clauses of length two.

In the same fashion as the clause gadgets, each variable gadget G_{x_i} admits two possible odd decycling matchings M as depicted in Fig. 4.9. We can see that the pair $\ell_i(3, b)$ and $\ell_i(3, b)$ has a different assignment for the other two pairs $\ell_i(k, b)$ and $\ell_i(k, b)$, $k \in \{1, 2\}$. Moreover, the last two pairs have the same assignment, as it can be seen in Fig. 4.9a and Fig. 4.9b. One more detail is that the unique possibilities for such pairs is such that $\ell_i(3, b)$ and $\ell_i(3, b)$ have opposite assignments if and only if the vertices $\ell_i(k, b)$ and $\ell_i(k, b)$ have the same assignment, $k \in \{1, 2\}$. Therefore we can associate the positive literal x_i to the pairs $\ell_i(k, b)$ and $\ell_i(k, b)$, $k \in \{1, 2\}$, while $\overline{x_i}$ can be represented by $\ell_i(3, b)$ and $\ell_i(3, b)$.

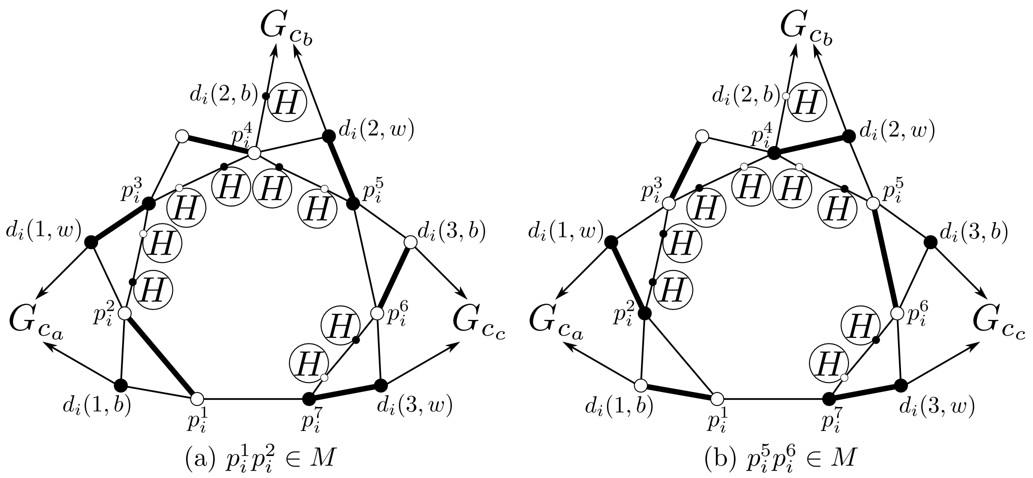


Figure 4.9: All possible configurations given by the removal of an odd decycling matching M, represented by the stressed edges, from a variable gadget G_{x_i}. The colors in the vertices represent the bipartition of $G_{x_i} - M$.

As observed above for clause gadgets, we can associate true value to the pair of vertices $\ell_j(k,w)$ and $\ell_j(k,b)$ with same color, $k \in \{1,2,3\}$. This implies that exactly one of them is true and, that is, exactly one literal of c_j has true value. Moreover, each variable gadget has two positive literals and a negative one, such that the positive and negative have opposite truth assignment.

Hence, if $G \in \mathcal{BM}$, then every clause gadget has exactly one true literal and every variable has a correct truth assignment, which implies that F is satisfiable. Conversely, if F is satisfiable, then each clause has exactly one true literal. Thus, for each clause c_j gadget we associate to pair of vertices corresponding to its true literal a same color. By Fig.4.8, there is an appropriate choice of an odd decycling matching for each true literal of c_j. Moreover, for each literal gadget G_{x_i} there is also an appropriate odd decycling matching for the choice of the true literal. This concludes the proof. □

Let G be the graph obtained by the construction in Theorem 4.17, next we show how to obtain a planar graph of maximum degree 4. Since the only vertices of degree 5 of G are those p_i^4 in the variable gadgets, we present a slightly modified variable gadget, which is depicted in Fig 4.10. In fact, we just modify the head graph. In Fig. 4.3 we can see that the vertex h_6 has degree 3, which allows us to use it to connect the variable gadget to the clause one. Fig. 4.11 shows the possible odd decycling matching of the modified variable gadget. Since such configurations are analogous to those of the original variable gadget, with respect to the vertex that connect to clause gadgets, we obtain our main result of this section.

Corollary 4.18 Odd Decycling Matching *is NP-complete even for 3-colorable planar graphs with maximum degree 4.*

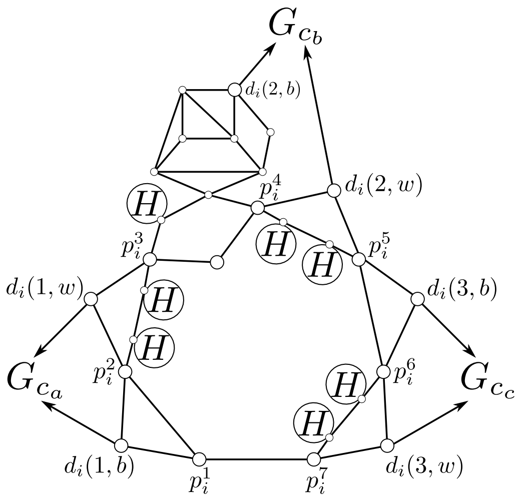


Figure 4.10: The modified variable gadget.

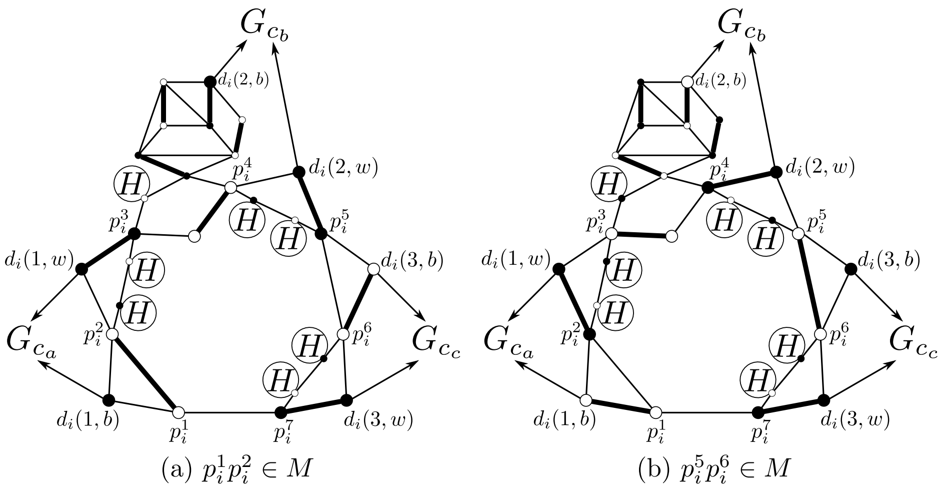


Figure 4.11: All possible configurations given by the removal of an odd decycling matching M from the modified variable gadget.

4.3 Polynomial Time Results

In this section we present our positive results about Odd Decycling Matching for some graph classes.

(Claw, Paw)-free Graphs

Now we consider graphs that have no induced subgraph isomorphic to the claw or to the paw. Let G be a connected (claw, paw)-free graph. We first prove that if $G \in \mathcal{BM}$, then the neighborhood of any vertex v of G has a small size.

Lemma 4.19 *If $G \in \mathcal{BM}$ is a claw-free graph, then $\Delta(G) \leq 5$.*

Proof. Suppose for a contradiction that G has a vertex v of degree at least 6, such that $\{v_1, v_2, v_3, v_4, v_5, v_6\} \subseteq N_G(v)$. Since G is claw-free, then either $G[N_G(v)]$ has

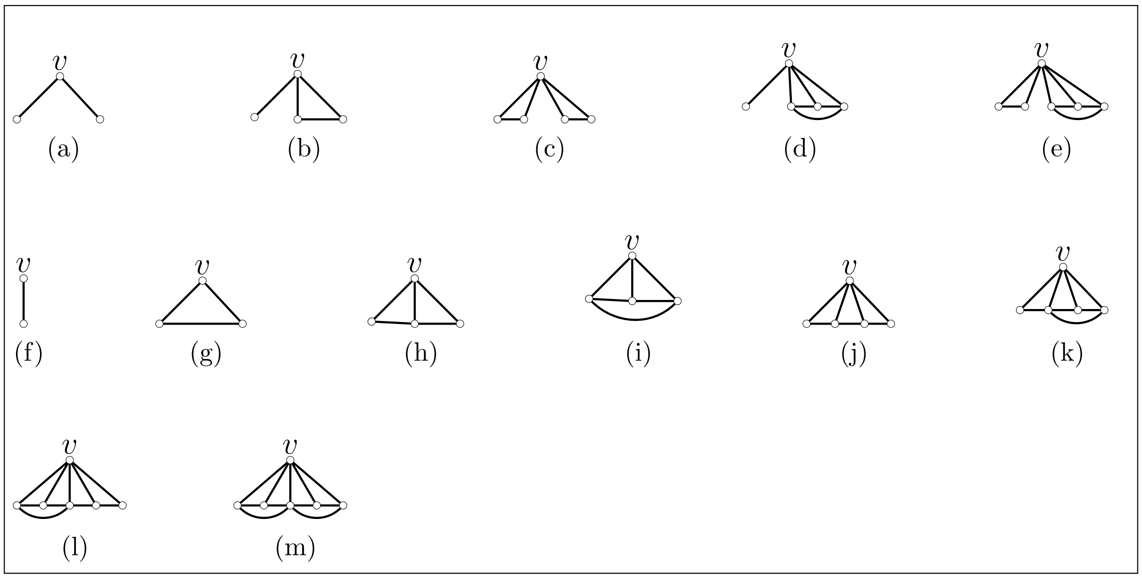


Figure 4.12: All possible neighborhoods of v of a claw-free graph $G \in \mathcal{BM}$.

exactly two connected components, which must be cliques, or itself is connected. In the first case, since G is K_5-free, it follows that each connected component of $G[N_G(v)]$ has size at most 3. Moreover, by Lemma 4.10(ii), if both have size at least 3, then $G \notin \mathcal{BM}$, a contradiction.

Now suppose that $G[N_G(v)]$ is connected. If $G[N_G(v)]$ has a $P_5 = abcde$ as a subgraph, then it is not difficult to see that the only odd decycling matching M of $G[\{v\} \cup \{a,b,c,d,e\}]$ should have the edges ab, vc and de. Moreover $ae \notin E(G)$, otherwise $G[\{v\} \cup \{a,b,c,d,e\}]$ has a W_5 as a subgraph, a contradiction by Lemma 4.10(iii). In this way, any other vertex $f \in N(v)$ can be adjacent only to vertex c, since all of the vertices in $\{v,a,b,c,d,e\}$ are matched by M. Hence $G[\{v\} \cup \{a,e,f\}]$ induces a claw in G, a contradiction.

On the other hand, necessarily $G[N_G(v)]$ has a P_3, otherwise there exist at least three connected components in $G[N_G(v)]$. By symmetry, suppose that v_1v_2 and $v_2v_3 \in E(G)$. By Lemma 4.10(ii) $G[\{v_4,v_5,v_6\}]$ has exactly one edge, say v_4v_5. Since $G[N_G(v)]$ is connected, the same has a path P of length at least 4 which connects v_6 to v_4 and v_5 by at least one vertex between v_1, v_2, and v_3. Since $G[N_G(v)]$ does not contain P_5, we can rename the vertices in $\{v_1,v_2,v_3,v_4,v_5,v_6\}$ such that $P = v_1v_2v_3v_4$. We know that $v_1v_4 \notin E(G)$, otherwise $P \cup \{v\}$ is a W_4, a contradiction by Lemma 4.10(iii). Hence v_5 and v_6 must be adjacent to at least one of v_1 and v_4, which creates a P_5 in $G[N_G(v)]$, a contradiction. $\square$

By Lemma 4.19, it is not hard to see that the possible neighborhoods of a vertex v of a claw-free graph $G \in \mathcal{BM}$ are depicted in Fig. 4.12.

It follows from Lemma 4.19 and of Fig. 4.12 that the only possibilities to the neighborhood of a vertex v in a $(claw, paw)$-free graph are as in Fig. 4.12a, Fig. 4.12f, Fig. 4.12g, Fig. 4.12h and Fig. 4.12i. In this way we can directly conclude the following lemma.

Lemma 4.20 *If $G \in \mathcal{BM}$ is a $(claw, paw)$-free graph then $\Delta(G) \leq 3$.*

Hence we can just apply Algorithm 4 to obtain a linear time algorithm for $(claw, paw)$-free graphs. Moreover, by Lemma 4.20 we can characterize the connected $(claw, paw)$-free graphs that admit an odd decycling matching.

Theorem 4.21 *If G is a connected $(claw, paw)$-free graph, then $G \in \mathcal{BM}$ if and only if G is isomorphic to a path, a cycle, a diamond or to a K_4.*

Proof. Let G be a connected $(claw, paw)$-free graph. Clearly, if G is isomorphic to a path, a cycle, a diamond or to a K_4, then $G \in \mathcal{BM}$.

Now consider $G \in \mathcal{BM}$. By Lemma 4.20 we know that does not exist any vertex of degree 4 in G.

If all of the vertices of G have degree 2, then G is isomorphic to a cycle.

If G is the trivial graph, then the theorem follows. Let v be a vertex of degree 1 in G. It follows that either G is a path of length at least one, or there is a vertex u of degree three. Consider u such that it is such a vertex closest to v in G. Let $w \in N_G(u)$ be in the path P from v to u and let u_1 and u_2 be its two neighbors except for w in G. Since $d_G(w) \leq 2$, we get that w is not adjacent neither to u_1, nor to u_2. Thus $u_1u_2 \in E(G)$, since G is claw-free. In this way $G[wuu_1u_2]$ is isomorphic to a paw, a contradiction. Hence G must be a path.

Finally suppose that G has a vertex v of degree 3 and that G is not isomorphic to a K_4. In this way it follows that $G[\{v\} \cup N_G(v)]$ is isomorphic to a diamond. Let $N_G(v) = \{v_1, v_2, v_3\}$, such that $d_G(v_2) = 3$. Since v and v_2 have degree 3, they cannot be adjacent to any other vertex. Suppose that v_1 has a neighbor $u \notin \{v, v_2\}$. Since uv, $uv_2 \notin E(G)$, then $G[\{u, v, v_1, v_2\}]$ is isomorphic to a paw, a contradiction. It follows in a similar way that $N_G(v_3) = \{v, v_2\}$. Hence G is isomorphic to a diamond, which concludes the proof. □

Graphs with Small Dominating Sets

We will show now that Odd Decycling Matching can be solved in polynomial time when the given graph G has a dominating set of size constant compared to G. Such a result generalizes some known graph classes, as fo example P_5-free graphs[31], since the graphs in $\mathcal{BM}$ do not admit K_5 as subgraph.

Theorem 4.22 *Let k be a positive integer. For a graph G whose domination number is at most k, it is possible to decide in polynomial time whether G has a matching M such that $G - M$ is bipartite, and to find such a matching if it exists.*

Proof. Let G be as in the statement. A dominating set of order at most k can be found in time $O(n^k)$. Let D be such a dominating set of G of order at most k. Let $\mathcal{P}_\mathcal{D}$ be the set of all bipartitions P_D of D into sets A_D and B_D, such that $D[A_D]$ and $D[B_D]$ do not have any vertex of degree 2. Note that $|\mathcal{P}_\mathcal{D}| = O(2^k)$.

Let $P_D \in \mathcal{P}_\mathcal{D}$ be a bipartition of D. We partition all of the other vertices of $G - D$ in such a way that P_D defines a bipartition of $G - M_D$, if one exists, where M_D is a matching that will be removed, given the choice of D. We do the following tests and operations for each vertex $v \in V(G) \setminus V(D)$:

- If $d_{A_D}(v) \geq 2$ and $d_{B_D}(v) \geq 2$, then P_D is not a valid partition;
- If $d_{A_D}(v) \geq 2$, then $A_D \leftarrow A_D \cup \{v\}$;
- If $d_{B_D}(v) \geq 2$, then $B_D \leftarrow B_D \cup \{v\}$.

Iteratively we allocate the vertices in $V(G) \setminus V(D)$ as described above into the respective sets A_D e B_D, or we stop if it is not possible to acquire a valid bipartition. After these operations, $V(G) \setminus V(A_D \cup B_D)$ can be partitioned into three sets as follows:

- $X = \{u \in V(G) \setminus V(A_D \cup B_D) : d_{A_D}(u) = 1 \text{ and } d_{B_D}(u) = 0\}$;
- $Y = \{u \in V(G) \setminus V(A_D \cup B_D) : d_{A_D}(u) = 0 \text{ and } d_{B_D}(u) = 1\}$;
- $Z = \{u \in V(G) \setminus V(A_D \cup B_D) : d_{A_D}(u) = 1 \text{ and } d_{B_D}(u) = 1\}$;

Since every vertex in $V(G) \setminus V(D)$ has a neighbor in D, it follows that the neighborhood of all the vertices of $X \cup Y \cup Z$ in $A_D \cup B_D$ is in D. In this way, we can make a choice of a matching M_D to be removed, such that all of the vertices of $X \cup Y \cup Z$ in $A_D \cup B_D$ are allocated in $A_D \cup B_D$ and $G - M_D$ must be bipartite. Note that there are $(n-k)^k$ possibilities of choices for M_D.

In this way, we can choose a dominating set D of order at most k and make all possible bipartitions P_D of $V(D)$ into sets A_D and B_D as previously described. For each one, we iteratively allocate the vertices of $V(G) \setminus V(D)$ into A_D and B_D in a unique way. After this operations, we test all possible matchings M_D with edges between vertices of D and those in $X \cup Y \cup Z$. We can see that in any case we do $O(n^k)$ operations, which concludes the proof. $\square$

Graphs with Only Triangles as Odd Cycles

We consider now a slightly general version of Odd Decycling Matching, where some edges are forbidden to be in any odd decycling matching.

Allowed Odd Decycling Matching (AODM)

Instance:	A graph G and a set F of edges of G.
Task:	Decide whether G has an odd decycling matching M that does not intersect F, and determine such a matching if it exists.

A matching M as in AODM is an *allowed odd decycling matching* of (G, F). Since (G, F) has an allowed odd decycling matching if and only if $(K, E(K) \cap F)$ has an allowed odd decycling matching for every component K of G, we may assume that G is connected. Moreover, note that if G has an allowed odd decycling matching, then $G \in \mathcal{BM}$, since all allowed odd decycling matching is an odd decycling matching of G. With this new notion, it follows the next result.

Theorem 4.23 AODM *can be solved in polynomial time for graphs with only triangles as odd cycles.*

Proof. Let G be a graph without odd cycles of size at least 5 and let F be a set of edges of G. As described above, we can consider G connected. Moreover, we can assume G as a bridge-free graph, that is, a graph whose all blocks have size at least 3.

Consider a block decomposition Π of G. Clearly we can assume that G has at least two blocks. Let B be a block of G that contains exactly one cut-vertex v, that is, B is a final block in Π. If $G[B]$ is bipartite, then clearly (G, F) has an allowed odd decycling matching if and only if $(G^{'}, F^{'})$ admits an allowed odd decycling matching, where $G^{'} = ((V(G) \setminus V(B)) \cup \{v\}, E(G) \setminus E(B))$ and $F^{'} = F \setminus E(B)$. Hence we can suppose that B has a triangle $v_1v_2v_3v_1$.

If $\{v_1v_2, v_1v_3\}$ is not a edge cut, then $G - \{v_1v_2, v_1v_3\}$ has a path P from v_1 to $\{v_2, v_3\}$. Consider P such a path of length at least 2 and such that it is a longest one. Moreover, consider v_2 as the first vertex reached by P between v_2 and v_3. In this way P must have the form v_1uv_2, otherwise either $G[V(P) \cup \{v_3\}]$ or $P \cup \{v_1v_2\}$ would contain an odd cycle of length at least 5, when P has either an even number of vertices or an odd number, respectively. It also follows that all path between v_1 and v_2, except v_1v_2, has length 2. Moreover, every path between v_1 and v_3, except v_1v_3, has length 2 and contains v_2.

If $uv_3 \in E(G)$, then $N_B(v_1) = \{u, v_2, v_3\}$, otherwise let $w \in N_B(v_1)$. Since $d_B(w) \geq 2$, we get that $N_B(w) = \{v_1, v_2\}$, otherwise would exist a path between v_1 and v_2 of length at least 3, a contradiction. However the cycle $v_1uv_3v_2wv_1$ has length 5, a contradiction. Thus $G[\{v_1, v_2, v_3, u\}]$ is isomorphic to a K_4 and $\{v_1v_2, v_1v_3, v_1u\}$ is an edge cut. Hence B is a block of G and, by symmetry, $v = v_1$. In this case we get that (G, F) has an allowed odd decycling matching if and only if $(G^{'}, F^{'})$ has an allowed odd decycling matching, where F does contain any matching of B of maximum size

$$G^{'} = ((V(G) \setminus V(B)) \cup \{v\}, E(G) \setminus E(B))$$

and

$$F^{'} = (F \setminus E(B)) \cup \{vx : x \in N_G(v) \setminus \{v_1, v_2, v_3\}\}.$$

If $uv_3 \notin E(G)$, then $\{zv_2, zv_1\}$ is an edge cut for every vertex $z \in N_B(v_2) \cap N_B(v_1)$. Furthermore, by symmetry, $N_B(v_2) \cap N_B(v_1)$ is an independent set. Thus $\{v_1, v_2\} \cup \{N_B(v_2) \cap N_B(v_1)\}$ is a block of G. If $v \in \{v_1, v_2\}$ and $|N_B(v_2) \cap N_B(v_1)| \geq 3$, then (G, F) has an allowed odd decycling matching is and only if $(G^{'}, F^{'})$ has an allowed odd decycling matching, where $v_1v_2 \notin F$ and

$$G^{'} = ((V(G) \setminus V(B)) \cup \{v\}, E(G) \setminus E(B))$$

and

$$F^{'} = (F \setminus E(B)) \cup \{vx : x \in N_G(v) \setminus V(B)\}.$$

If $v \in \{v_1, v_2\}$ and $|N_B(v_2) \cap N_B(v_1)| = 2$, then (G, F) has an allowed odd decycling matching if and only if $(G^{'}, F^{'})$ has an allowed odd decycling matching, where $v_1v_2 \notin F$, or $F \not\supseteq \{v_1u, v_2v_3\}$, or $F \not\supseteq \{v_1v_3, v_2u\}$, and

$$G^{'} = ((V(G) \setminus V(B)) \cup \{v\}, E(G) \setminus E(B)) \text{, and } F^{'} = (F \setminus E(B)) \cup \{vx : x \in N_G(v) \setminus V(B)\},$$

If $v \notin \{v_1, v_2\}$, then (G, F) has an allowed odd decycling matching if and only if $(G^{'}, F^{'})$ has an allowed odd decycling matching, where $v_1v_2 \notin F$ and

$$G^{'} = ((V(G) \setminus V(B)) \cup \{v\}, E(G) \setminus E(B))$$

and

$$F^{'} = (F \setminus E(B)).$$

Finally it remains the case that $G[\{v_1, v_2, v_3\}]$ is a block and, by symmetry, suppose $v = v_1$. Thus (G, F) has an allowed odd decycling matching if and only if $(G^{'}, F^{'})$ has an allowed odd decycling matching, where $v_2v_3 \notin F$ and

$$G^{'} = ((V(G) \setminus V(B)) \cup \{v\}, E(G) \setminus E(B))$$

and

$$F^{'} = F \setminus E(B),$$

or $v_1v_2 \notin F$ or, $v_1v_3 \notin F$, and

$$G^{'} = ((V(G) \setminus V(B)) \cup \{v\}, E(G) \setminus E(B))$$

and

$$F^{'} = (F \setminus E(B)) \cup \{vx : x \in N_G(v) \setminus V(B)\}.$$

This concludes the proof. □

4.3.1 Fixed-Parameter Tractability

In this section, we consider the parameterized complexity of Odd Decycling Matching, and present an analysis of its complexity when parameterized by some classical parameters.

Definition 4.24 *The clique-width of a graph G, denoted by $cwd(G)$, is defined as the minimum number of labels needed to construct G, using the following four operations [26]:*

1. *Create a single vertex v with an integer label ℓ (denoted by $\ell(v)$);*
2. *Disjoint union of two graphs (i.e. co-join) (denoted by $\oplus$);*
3. *Join by an edge every vertex labeled i to every vertex labeled j for $i \neq j$ (denoted by $\eta(i, j)$);*
4. *Relabeling all vertices with label i by label j (denoted by $\rho(i, j)$).*

Some graph classes with bounded clique-width include cographs [26], distance-hereditary graphs [85], graphs with bounded neighborhood diversity [100], and graphs with bounded tree-width such as forests, pseudoforests, cactus graphs, series-parallel graphs, outerplanar graphs, Halin graphs, Apollonian networks, and control flow graphs [21, 25, 134].

Courcelle, Makowsky and Rotics [48] stated that for any graph G with clique-width bounded by a constant k, and for each graph property Π that can be formulated in a *monadic second order logic* ($MSOL_1$), there is a $f(cwd(G)).n$ algorithm that decides if G satisfies Π (cf. [47–51]). In this monadic second-order graph logic known as $MSOL_1$, the graph is described by a set of vertices V and a binary adjacency relation $edge(.,.)$, and the graph property in question may be defined in terms of sets of vertices of the given graph, but not in terms of sets of edges.

Using Courcelle, Makowsky and Rotics's meta-theorem based on monadic second order logic [48], in order to show the fixed-parameter tractability of Odd Decycling Matching when parameterized by clique-width, it remains to show that the related property is $MSOL_1$-expressible.

Theorem 4.25 ODD DECYCLING MATCHING *is fixed-parameter tractable when parameterized by the clique-width.*

Proof. Remind that the problem of determining whether G has an odd decycling matching is equivalent to determine whether G admits an $(1,1)$-coloring, which is a 2-coloring of $V(G)$ in which each color class induces a graph of maximum degree at most 1 (cf. [103]). Thus, it is enough to show that the property "G has an $(1,1)$-coloring" is $MSOL_1$-expressible.

We construct a formula $\varphi(G)$ such that $G \in \mathcal{BM} \Leftrightarrow \varphi(G)$ as follows:

$$\begin{aligned}
\exists\, S_1, S_2 \subseteq V(G)\ : & (S_1 \cap S_2 = \emptyset) \wedge \\
& (S_1 \cup S_2 = V(G)) \wedge \\
& (\forall\, v_1 \in S_1[\, \nexists\, u_1, w_1 \in S_1 : (u_1 \neq w_1) \wedge edge(u_1, v_1) \wedge edge(w_1, v_1)]) \wedge \\
& (\forall\, v_2 \in S_2[\, \nexists\, u_2, w_2 \in S_2 : (u_2 \neq w_2) \wedge edge(u_2, v_2) \wedge edge(w_2, v_2)])
\end{aligned}$$

□

From Theorem 4.25 we can solve several interesting classes of graphs in polynomial time, as for example some subclasses of planar graphs such as outerplanar graphs, Halin graphs and Apollonian networks. In addition, since clique-width generalizes several graph parameters [100], we have the following corollary.

Corollary 4.26 ODD DECYCLING MATCHING *is fixed-parameter tractable when parameterized by the following parameters:*

- *neighborhood diversity;*
- *treewidth;*
- *pathwidth;*
- *feedback vertex set;*
- *vertex cover.*

Courcelle's theorem is a good classification tool, however it does not provide a precise running time bound. Next result shows the exact upper bound for ODD DECYCLING MATCHING parameterized by $\beta(G)$.

Theorem 4.27 ODD DECYCLING MATCHING *admits a* $2^{O(\beta(G))}.n$ *algorithm.*

Proof. Let S be a vertex cover of G such that $|S| = \beta(G)$. The algorithm follows in a similar way to Algorithm 4.22. Let $\mathcal{P}_S$ be the set of all bipartitions P_S of S

into sets A_S and B_S, such that $S[A_S]$ and $S[B_S]$ do not have any vertex of degree 2. Note that $|\mathcal{P}_S| = O(2^k)$.

For each $P_S \in \mathcal{P}_S$, we will check if an odd decycling matching of G can be obtained from P_S by applying the following operations:

For each vertex $v \in V(G) \setminus V(S)$ do

- If $d_{A_S}(v) \geq 2$ and $d_{B_S}(v) \geq 2$, then P_S is not a valid partial partition;
- If $d_{A_S}(v) \geq 2$, then $A_S \leftarrow A_S \cup \{v\}$;
- If $d_{B_S}(v) \geq 2$, then $B_S \leftarrow B_S \cup \{v\}$.

After that, if for all vertices the first condition is not true, then $V(G) \setminus V(A_S \cup B_S)$ can be partitioned into three sets:

- $X = \{u \in V(G) \setminus V(A_S \cup B_S) : d_{A_S}(u) = 1 \text{ and } d_{B_S}(u) = 0\}$;
- $Y = \{u \in V(G) \setminus V(A_S \cup B_S) : d_{A_S}(u) = 0 \text{ and } d_{B_S}(u) = 1\}$;
- $Z = \{u \in V(G) \setminus V(A_S \cup B_S) : d_{A_S}(u) = 1 \text{ and } d_{B_S}(u) = 1\}$;

Since $V(G) \setminus V(S)$ is an independent set, it follows that all edges of vertices in $X \cup Y$ can remain in the graph G. For each $z \in Z$, denote by $a_z \in A_S$ and $b_z \in B_S$ the neighbors of z in G.

Now, we apply a bounded search tree algorithm. While $G[A_S]$ and $G[B_S]$ have both maximum degree equal to one, and $Z \neq \emptyset$ do. Remove a vertex $z \in Z$ and apply recursively the algorithm for the following cases:

1. z is added to A_S, and all vertices in $Z \cap N(a_z)$ is added to B_S.
2. z is added to B_S, and all vertices in $Z \cap N(b_z)$ is added to A_S.

Note that the search tree T has height equals $\beta(G) + 1$. Finally, if T has a leaf representing a configuration with $G[A_S]$ and $G[B_S]$ having both maximum degree equal to one, and $Z = \emptyset$ then G has an odd decycling matching. □

ow we analyze the parameterized complexity of Allowed Odd Decycling Matching considering the neighborhood diversity number, $nd(G)$, as parameter.

Definition 4.28 *A graph $G(V, E)$ has neighborhood diversity $nd(G) = t$ if we can partition V into t sets $V_1, \dots, V_t$ such that, for every $v \in V$ and all $i \in 1, \dots, t$, either v is adjacent to every vertex in V_i or it is adjacent to none of them. Note that each part V_i of G is either a clique or an independent set.*

The neighborhood diversity parameter is a natural generalization of the vertex cover number. In 2012, Lampis [100] showed that for every graph G we have $nd(G) \leq 2^{vc(G)} + vc(G)$. The optimal neighborhood diversity decomposition of a graph G can be computed in $\mathcal{O}(n^3)$ time [100].

Theorem 4.29 Allowed Odd Decycling Matching *admits a kernel with at most* $2.nd(G)$ *vertices when parameterized by neighborhood diversity number.*

Proof. Given an instance (G, F) of Allowed Odd Decycling Matching such that G is a graph and $F \subseteq E(G)$ a set of forbidden edges. The kernelization algorithm consists on applying the following reduction rules:

1. *If* G *contains a* K_5*, then* G *has no allowed odd decycling matching; otherwise*
2. *If a part* V_i *induces a* K_3 *and exist two vertices in* $V(G) \setminus V_i$ *adjacent to* V_i*, then* G *has no allowed odd decycling matching; otherwise*
3. *If a subgraph of* G *induces either a* K_3 *or a* K_4 *and does not admit an allowed odd decycling matching, then* G *has no allowed odd decycling matching; otherwise*
4. *Remove all parts isomorphic to a* K_4*;*
5. *Remove all isolated parts isomorphic to a* K_3*;*
6. *If* V_i *is a part that induces a* K_3 *and* $v \in V(G) \setminus V_i$ *is adjacent to* V_i *(note that* $\{v\}$ *is a part), then remove* V_i *and* $F \leftarrow F \cup \{uv : u \in N_G(v) \setminus V_i\}$*;*
7. *If a part* V_i *induces an independent set of size at least 3, then contract it into a single vertex* v_i *(without parallel edges) and forbids all of its incident edges;*

It is easy to see that all reduction rules can be applied in polynomial time, and after applying them any remaining part has size at most two. As the resulting graph G' has $nd(G') \leq nd(G)$ then $|V(G')| \leq 2.nd(G)$. Thus, it remains to prove that the application of each reduction rule is correct. As K_5 and $K_5 - e$ are forbidden subgraphs, and any odd decycling matching of a K_4 is a perfect matching then rules 1,2,3,4,5 and 6 can be applied in such an order. Finally, the correctness of rule 7 follows from the following facts: (i) if G' has an allowed odd decycling matching, then G has also an allowed odd decycling matching, because bipartite graph class is closed under the operation of replacing vertices by a set of false twins, which have the same neighborhood as the replaced vertex; (ii) if G' does not admit an allowed odd decycling matching then G also does not admit an allowed odd decycling matching, because if a contracted single vertex v_i is in an odd cycle in G', then even replacing v_i by V_i ($|V_i| \geq 3$) and removing some incident edges of V_i which form a matching, some vertex of V_i remains to an odd cycle. □

Chapter 5

Conclusion

We have presented our results obtained during the doctorate. The sequence of the chapters follows the order of obtained results and submissions. We have studied five different problems, with some similar characteristics, that generate a number of submissions, almost of them without answer until now.

We first studied the f-reversible processes and their natural parameters, such as the period and transient lengths. We have obtained a tight upper bound for the transient length and show that the period length is also limited by two as well as the threshold networks. We also characterized the trees that reach the maximum transient length for 2-reversible processes, that we prove to be equals $n-3$. Finally we prove that determining the minimum f-conversion set is NP-hard even for bipartite graphs with bounded degree.

At the same time we have worked on the AND/OR-convexity, where we proved many NP-hardness results and polynomial time algorithms for the different parameters considered, that were convexity number, hull number, interval number, and Charathéodory number.

The generalized threshold processes are considered right after, where its motivation comes from processes in distributed computing, such as the `xor`-circuits. We have proved several NP-hardness results even for simple classes as the complete graphs. However, we present efficient algorithms for trees.

The previous problems use notions of convexity and iterative processes. In Chapter 4 we start the study of editing problems on graphs. We consider two cases where it is desired to know whether a given graph G admits a matching whose removal generates a forest or a bipartite graph. We have proved that both versions are NP-hard and present several polynomial time algorithms for some specific graph classes. We also considered parameterized results about the second version.

All results of this thesis are submitted or published.

5.1 Open Problems

"It's the job that's never started as it takes longest to finish."
(Sam)

— J.R.R. Tolkien, *The Fellowship of the Ring*

5.1.1 f-Reversible Processes

The potential function gave the necessary interpretation in other to obtain all initial configurations on 2-reversible processes on trees reaching at least $n-3$ time steps, where its value is tight when $n \geq 4$. Determining such configurations based on only G and f is a no trivial work. This can be observed by the work of Oliveira, Barbosa and Protti [116], in which determining if there exists a predecessor configuration of a given one is NP-complete even for bipartite graphs. In this way, we left two problems regarding the maximum transient length:

- To characterize all initial configurations reaching the maximum transient length in terms of only G and f;
- To count the number of such configurations.

We also have studied the problem of determining the minimum size $r_f(G)$ of a vertex subset that allows f-reversible processes to uplift. We have shown that determining if $r_f(G)$ is at most a constant $q > 0$ is NP-complete for bipartite graphs with maximum degree 3 and $\text{Im}_f = \{1, 2, 3\}$. We also have proved that $\beta(G) \leq r_f(G) \leq \beta(G) + 1$, when $f(v) = d(v)$ for all vertices. In fact, determining $r_f(G)$ seems a no trivial job even for simple graph classes as paths and cycles [59]. In addition to this open problem, we propose the following:

- To determine a lower and an upper bound on $r_f(G)$;
- To find an approximative algorithm to compute $r_f(G)$.

5.1.2 AND/OR-Convexity

We leave open the study of other convexity parameters, such as the Radon number, the rank, and the Helly number. It is interesting to know what is the complexity of computing them in the AND-OR model, and what are their meanings in this model.

5.1.3 Generalized Threshold Process

Many questions about this new problem can be done. We left some of them here.

What is the complexity of the non-relaxed τ-hull number for trees? Are there non-trivial bounds on the relaxed or non-relaxed τ-hull number?

It seems interesting to study a 'total' version of τ-interval sets, where S is a *total τ-interval set* of a graph G with threshold function τ if $N_G(u) \cap S \in \tau(u)$ for every vertex u of G, and not only for every vertex u in $V(G) \setminus S$. It is easy to see that deciding the existence of a total $\oplus$-interval set for a given graph is equivalent to a suitable system of n linear equalities over $\mathbb{F}_2$.

Let V be a set of order n. For which vectors $(t_0, t_1, \ldots, t_n)$ does there exist a set system $\mathcal{T} \subseteq 2^V$ with $\forall X \in \mathcal{T} : \forall Y \subseteq V : X \subseteq Y \Rightarrow Y \in \mathcal{T}$ such that $t_i = \left|\mathcal{T} \cap \binom{V}{i}\right|$ for every i?

5.1.4 Decycling with a Matching

we only study a special case of the more general problem of destroying all cycles by removing edges under the restriction that the graph formed by the removed edges has bounded maximum degree. This problem can certainly be considered more generally.

Another class of graphs where Allowed Decycling Matching might be solvable efficiently are chordal bipartite graphs. Their guaranteed density is close to the threshold from Lemma 4.1(ii), which should imply strong restrictions on the block structure of chordal bipartite graphs with a decycling matching.

5.1.5 Odd Decycling with a Matching

The polynomial time results obtained here can clearly be extended to other graph classes, as well as characterizations of such graph classes that have an odd decycling matching. Such polynomial time algorithms must be applicable to sparse graphs, since they are even subgraph of K_5 free, such as the W_4.

We found a good dichotomy related to the NP-completeness of Odd Decycling Matching, where it is know that subcubic graphs have a linear time algorithm and it is NP-complete even for planar graphs with degree at most 4.

As we saw, the minimization version have received attention in the literature, where we can see that Odd Decycling Matching coincides to MAX-CUT when restricted to subcubic graphs. The ideas presented in this work can be used to obtain new combinatorial approximative algorithms with a better approximation factor than the best one know, which is based on semi-defined programming [18].

Interesting properties regarding the chromatic number of graphs in $\mathcal{BM}$ can be proposed. For example, which graphs $G \in \mathcal{BM}$ are such that $\chi(G - M) \leq \chi(G)$, for an odd decycling matching M of G? Moreover, what is the maximum size of the gap between $\chi(G - M)$ and $\chi(G)$?

Bibliography

[1] Abdullah, A., 1992, "On graph bipartization". In: *Circuits and Systems, 1992. ISCAS '92. Proceedings., 1992 IEEE International Symposium on*, v. 4, pp. 1847–1850, May.

[2] Ackerman, E., Ben-Zwi, O., Wolfovitz, G., 2010, "Combinatorial model and bounds for target set selection", *Theoretical Computer Science*, v. 411, pp. 4017–4022.

[3] Agur, Z., 1991, "Fixed points of majority rule cellular automata with application to plasticity and precision of the immune system", *Complex Syst.*, v. 5, pp. 351–357.

[4] Allouche, J.-P., Courbage, M., Skordev, G., 2001, "Notes on cellular automata", *Complex Syst.*, v. 3, pp. 213–244.

[5] Alon, N., Stav, U., 2009, "Hardness of edge-modification problems", *Theoretical Computer Science*, v. 410, n. 47–49, pp. 4920 – 4927.

[6] Aral, S., Walker, D., 2012, "Identifying Influential and Susceptible Members of Social Networks", *Science*, v. 337, n. 6092, pp. 337–341.

[7] Aral, S., Muchnik, L., Sundararajan, A., 2009, "Distinguishing influence-based contagion from homophily-driven diffusion in dynamic networks", *Proc. Natl. Acad. Sci.*, v. 106, n. 51, pp. 21544–21549.

[8] Araujo, J., Campos, V., Giroire, F., et al., 2013, "On the Hull Number of Some Graph Classes", *Theor. Comput. Sci.*, v. 475, pp. 1–12.

[9] Balister, P., Bollobás, B., Johnson, J. R., et al., 2010, "Random Majority Percolation", *Random Struct. Algorithms*, v. 36, n. 3, pp. 315–340.

[10] Banda, P., 2011, "Cellular Automata Evolution of Leader Election". In: *Adv. Artif. Life. Darwin Meets von Neumann*, v. 5778, *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, pp. 310–317.

[11] Bandelt, H.-J., Mulder, H., 1986, "Distance-hereditary graphs", *Journal of Combinatorial Theory, Series B*, v. 41, pp. 182–208.

[12] Barbosa, V. C., 2002, "The Combinatorics of Resource Sharing". In: Corrêa, R., Dutra, I., Fiallos, M., et al. (Eds.), *Models for Parallel and Distributed Computation*, v. 67, *Appl. Opt.*, Springer US, pp. 27–52.

[13] Barbosa, V. C., 1996, *An Introduction to Distributed Algorithms.* MIT Press.

[14] Barbosa, V. C., Benevides, M. R. F., 1998, *A Graph-Theoretic Characterization of AND-OR Deadlocks.* Relatório técnico, Federal University of Rio de Janeiro.

[15] Barnett, J., Verma, T., 1994, "Intelligent reliability analysis". In: *Proceedings of the Tenth Conference on Artificial Intelligence for Applications, 1994.*, pp. 428–433. IEEE, Mar.

[16] Barrett, C. L., Hunt III, H. B., Marathe, M. V., et al., 2006, "Complexity of reachability problems for finite discrete dynamical systems", *J. of Computer Syst. Sci.*, v. 72, n. 8, pp. 1317 – 1345.

[17] Bau, S., Beineke, L., 2002, "The decycling number of graphs", *Australasian Journal of Combinatorics*, v. 25, pp. 285–298.

[18] Bazgan, C., Tuza, Z., 2008, "Combinatorial 5/6-approximation of Max Cut in graphs of maximum degree 3", *Journal of Discrete Algorithms*, v. 6, n. 3, pp. 510 – 519.

[19] Beineke, L., Vandell, R., 1997, "Decycling graphs", *Journal of Graph Theory*, v. 25, pp. 59–77.

[20] Bertossi, A. A., 1984, "Dominating sets for split and bipartite graphs", *Inform. Process. Lett.*, v. 19, n. 1, pp. 37–40.

[21] Bodlaender, H. L., 1998, "A partial k-arboretum of graphs with bounded treewidth", *Theoretical computer science*, v. 209, n. 1, pp. 1–45.

[22] Bondy, A., Murty., U., 2008, *Graph Theory.* Graduate Texts in Mathematics. Spring-Verlag Press.

[23] Bondy, J. A., Locke, S. C., 1986, "Largest bipartite subgraphs in triangle-free graphs with maximum degree three", *Journal of Graph Theory*, v. 10, n. 4, pp. 477–504.

[24] Borodin, O., Kostochka, A., Yancey, M., 2013, "On 1-improper 2-coloring of sparse graphs", *Discrete Mathematics*, v. 313, n. 22, pp. 2638–2649.

[25] Brandstädt, A., Spinrad, J. P., others, 1999, *Graph classes: a survey*, v. 3. Siam.

[26] Brandstädt, A., Dragan, F. F., Le, H.-O., et al., 2005, "New graph classes of bounded clique-width", *Theory of Computing Systems*, v. 38, n. 5, pp. 623–645.

[27] Brzezinski, J., Helary, J.-M., Raynal, M., et al., 1995, "Deadlock models and a general algorithm for distributed deadlock detection", *J. Parallel. Distr. Com.*, v. 31, n. 2, pp. 112–125.

[28] Burzyn, P., Bonomo, F., Durán, G., 2006, "NP-completeness results for edge modification problems", *Discrete Applied Mathematics*, v. 154, n. 13, pp. 1824 – 1844.

[29] Cáceres, J., Hernando, C., Mora, M., et al., 2006, "On geodetic sets formed by boundary vertices", *Discrete Math.*, v. 306, n. 2, pp. 188–198.

[30] Calder, J., 1971, "Some elementary properties of interval convexities", *Journal of the London Mathematical Society*, v. 3, pp. 422–428.

[31] Camby, E., Schaudt, O., 2016, "A New Characterization of P_k-Free Graphs", *Algorithmica*, v. 75, n. 1, pp. 205–217.

[32] Campos, V., Sampaio, R. M., Silva, A., et al., 2015, "Graphs with few P_4's under the convexity of paths of order three", *Discrete Appl. Math.*, v. 192, n. 0, pp. 28–39.

[33] Cao, T., Sanderson, A. C., 1998, "AND/OR Net Representation for Robotic Task Sequence Planning", *IEEE T. Syst. Man. CY. C.*, v. 28, n. 2, pp. 204–218.

[34] Centeno, C., Rautenbach, D., 2015, "Remarks on dynamic monopolies with given average thresholds", *Discussiones Mathematicae Graph Theory*, v. 35, pp. 133–140.

[35] Chandy, K. M., Lamport, L., 1985, "Distributed snapshots: determining global states of distributed systems", *ACM T. Comput. Syst.*, v. 3, n. 1, pp. 63–75.

[36] Chen, N., 2009, "On the approximability of influence in social networks", *SIAM J. Discret. Math.*, v. 23, pp. 1400–1415.

[37] Chlebík, M., Chlebíková, J., 2008, "Approximation hardness of dominating set problems in bounded degree graphs", *Information and Computation*, v. 206, pp. 1264–1275.

[38] Choi, H.-A., Nakajima, K., Rim, C. S., 1989, "Graph Bipartization and via minimization", *SIAM Journal on Discrete Mathematics*, v. 2, n. 1, pp. 38–47.

[39] Cicalese, F., Milanič, M., Vaccaro, U., 2013, "On the approximability and exact algorithms for vector domination and related problems in graphs", *Discrete Applied Mathematics*, v. 161.6, pp. 750–767.

[40] Cicalese, F., Cordasco, G., Gargano, L., et al., 2014, "Latency-bounded target set selection in social networks", *Theoretical Computer Science*, v. 535, pp. 1–15.

[41] Cicalese, F., Cordasco, G., Gargano, L., et al., 2015, "Spread of influence in weighted networks under time and budget constraints", *Theoretical Computer Science*, v. 586, pp. 40–58.

[42] Cockayne, E., Goodman, S., Hedetniemi, S., 1975, "A linear algorithm for the domination number of a tree", *Inform. Process. Lett.*, v. 4, n. 2, pp. 41–44.

[43] Cockayne, E., Gamble, B., Shepherd, B., 1985, "An upper bound for the k-domination number of a graph", *Journal of Graph Theory*, v. 9, pp. 533–534.

[44] Coelho, E. M., Dourado, M. C., Rautenbach, D., et al., 2014, "The Carathéodory number of the convexity of chordal graphs", *Discrete Appl. Math.*, v. 172, n. 0, pp. 104–108.

[45] Conradi, R., Westfechtel, B., 1998, "Version Models for Software Configuration Management", *ACM Comput. Surv.*, v. 30, n. 2 (jun.), pp. 232–282.

[46] Cormen, T. H., Leiserson, C. E., Rivest, R. L., et al., 2009, *Introduction to Algorithms, Third Edition.* The MIT Press. ISBN: 0262033844, 9780262033848.

[47] Courcelle, B., Mosbah, M., 1993, "Monadic second-order evaluations on tree-decomposable graphs", *Theoretical Computer Science*, v. 109, n. 1–2, pp. 49 – 82.

[48] Courcelle, B., Makowsky, J. A., Rotics, U., 2000, "Linear Time Solvable Optimization Problems on Graphs of Bounded Clique-Width", *Theory of Computing Systems*, v. 33, n. 2, pp. 125–150.

[49] Courcelle, B., 1990, "The monadic second-order logic of graphs. I. Recognizable sets of finite graphs", *Information and Computation*, v. 85, n. 1, pp. 12 – 75.

[50] Courcelle, B., 1997, "The Expression of Graph Properties and Graph Transformations in Monadic Second-Order Logic." *Handbook of Graph Grammars*, v. 1, pp. 313–400.

[51] Courcelle, B., Engelfriet, J., 2011, *Graph structure and monadic second-order logic*, v. 138. Citeseer.

[52] da F. Ramos, I., dos Santos, V. F., Szwarcfiter, J. L., 2014, "Complexity aspects of the computation of the rank of a graph", *Discrete Math. & Theor. Comput. Sci.*, v. 16, n. 2, pp. 73–86.

[53] Dawes, R., 1989, "Minimum odd neighbourhood covers for trees", *Lect. Notes Comput. Sci.*, v. 507, pp. 161–169.

[54] de Mello, L. S. H., Sanderson, A. C., 1991, "A Correct and Complete Algorithm for the Generation of Mechanical Assembly Sequences", *IEEE T. Robotic Autom.*, v. 7, n. 2, pp. 228–240.

[55] Dinur, I., Steurer, D., 2014, "Analytical approach to parallel repetition". In: *Proceedings of the 46th Annual ACM Symposium on Theory of Computing*, pp. 624–633.

[56] Domingos, P., Richardson, M., 2001, "Random majority percolation", *Proc. 7th ACM SIGKDD Int. Conf. Knowl. Discov. Data Min.*, pp. 57–67.

[57] dos S. Souza, U., 2014, *Multivariate Investigation of NP-Hard Problems: Boundaries Between Parameterized Tractability and Intractability.* Graduate theses, Fluminense Federal University, Brazil.

[58] dos S. Souza, U., Protti, F., da Silva, M. D., 2013, "Revisiting the complexity of and/or graph solution", *J. Compu. Syst. Sci.*, v. 79, n. 7, pp. 1156–1163.

[59] Dourado, M., Penso, L., Rautenbach, D., et al., 2012, "Reversible iterative graph processes", *Theor. Comput. Sci.*, v. 460, pp. 16–25.

[60] Dourado, M., Rautenbach, D., dos Santos, V., et al., 2013, "On the Carathéodory number of interval and graph convexities", *Theoretical Computer Science*, v. 520, pp. 127–135.

[61] Dourado, M. C., Gimbel, J. G., Kratochvíl, J., et al., 2009, "On the computation of the hull number of a graph", *Disc. Math.*, v. 309, n. 18, pp. 5668–5674.

[62] Dourado, M. C., Protti, F., Szwarcfiter, J. L., 2010, "Complexity results related to monophonic convexity", *Discrete Appl. Math.*, v. 158, n. 12, pp. 1268–1274.

[63] Dourado, M. C., Rautenbach, D., dos Santos, V. F., et al., 2012, "Characterization and recognition of Radon-independent sets in split graphs", *Inform. Process. Lett.*, v. 112, n. 24, pp. 948–952.

[64] Dreyer Jr., P. A. ., Roberts, F. S., 2009, "Irreversible k-threshold processes: Graph-theoretical threshold models of the spread of disease and of opinion", *Discret. Appl. Math.*, v. 157, n. 7, pp. 1615 – 1627.

[65] Dreyer Junior, P. A., 2000, *Application and Variations of Domination in Graphs.* Ph.D. dissertation, The State University of New Jersey, New Brunswick, NJ.

[66] Erdős, P., Saks, M., Sós, V., 1986, "Maximum induced trees in graphs", *Journal Combinatorial Theory, Series B*, v. 41, pp. 61–79.

[67] Erdös, P., 1965, "On some extremal problems in graph theory", *Israel Journal of Mathematics*, v. 3, n. 2, pp. 113–116.

[68] Escoffier, B., Paschos, V. T., 2006, "Completeness in approximation classes beyond {APX}", *Theor. Comput. Sci.*, v. 359, n. 1–3, pp. 369 – 377.

[69] Favaron, O., Hansberg, A., Volkmann, L., 2008, "On k-domination and minimum degree in graphs", *Journal of Graph Theory*, v. 57, pp. 33–40.

[70] Feige, U., 1998, "A threshold of $\ln n$ for approximating set cover", *Journal of the ACM*, v. 45, pp. 634–652.

[71] Flocchini, P., Geurts, F., Santoro, N., 2001, "Optimal irreversible dynamos in chordal rings", *Discret. Appl. Math.*, v. 113, n. 1, pp. 23 – 42. Selected Papers: 12th Workshop on Graph-Theoretic Concepts in Computer Science.

[72] Flocchini, P., Královič, R., Ružička, P., et al., 2003, "On time versus size for monotone dynamic monopolies in regular topologies", *J. Discret. Algorithms*, v. 1, n. 2, pp. 129 – 150. {SIROCCO} 2000.

[73] French Jr., P., J. R., 1956, “A formal theory of social power”, *Psychol. Rev.*, v. 63, pp. 181–194.

[74] Furmańczyk, H., Kubale, M., Radziszowski, S., 2015, “On bipartization of cubic graphs by removal of an independent set”, *Discrete Applied Mathematics.* In Press, Corrected Proof.

[75] Garey, M. R., Johnson, D. S., 1979, *Computers and Intractability: A Guide to the Theory of NP-Completeness.* San Francisco, CA, W. H. Freeman & Co.

[76] Garey, M., Johnson, D., Stockmeyer, L., 1976, “Some simplified NP-complete graph problems”, *Theoretical Computer Science*, v. 1, n. 3, pp. 237 – 267.

[77] Garey, M., Johnson, D., Tarjan, R., 1976, “The planar Hamiltonian circuit problem is NP-complete”, *SIAM Journal on Computing*, v. 5, pp. 704–714.

[78] Gargano, L., Hell, P., Peters, J. G., et al., 2015, “Influence diffusion in social networks under time window constraints”, *Theor. Comput. Sci.*, v. 584, pp. 53 – 66. Special Issue on Structural Information and Communication Complexity.

[79] Gassner, E., Hatzl, J., 2008, “A parity domination problem in graphs with bounded treewidth and distance-hereditary graphs”, *Computing*, v. 82.2-3, pp. 171–187.

[80] Gentner, M., Rautenbach, D., 2015, “Feedback vertex sets in cubic multigraphs”, *Discrete Mathematics*, v. 338, n. 12, pp. 2179–2185.

[81] Goles, E., Olivos, J., 1981, “Comportement périodique des fonctions á seuil binaires et applications”, *Discrete Appl. Math.*, v. 3, n. 2, pp. 93 – 105.

[82] Goles, E., Olivos, J., 1985, “Periodic behavior of binary threshold functions and applications”, *Discrete Appl. Math.*, v. 3, pp. 93–105.

[83] Goles, E., Martínez, S., 2013, *Neural and automata networks: dynamical behavior and applications*, v. 58, *Mathematics and Its Applications.* Springer Netherlands.

[84] Goles-Chacc, E., Fogelman-Soulie, F., Pellegrin, D., 1985, “Decreasing energy functions as a tool for studying threshold networks”, *Discret. Appl. Math.*, v. 12, n. 3, pp. 261–277.

[85] Golumbic, M. C., Rotics, U., 2000, "On the clique-width of some perfect graph classes", *International Journal of Foundations of Computer Science*, v. 11, n. 03, pp. 423–443.

[86] Guillemot, S., Havet, F., Paul, C., et al., 2012, "On the (Non-)Existence of Polynomial Kernels for P-Free Edge Modification Problems", *Algorithmica*, v. 65, n. 4, pp. 900–926.

[87] Hassin, Y., Peleg, D., 2001, "Distributed Probabilistic Polling and Applications to Proportionate Agreement", *Inform. Comput.*, v. 171, n. 2, pp. 248 – 268.

[88] Haynes, T., Hedetniemi, S., Slater, P., 1998, *Fundamentals of Domination in Graphs.* Chapman Hall/CRC Pure and Applied Mathematics. Cleveland, Ohio, USA, CRC Press.

[89] Hopcroft, J., Tarjan, R., 1974, "Efficient Planarity Testing", *J. ACM*, v. 21, n. 4 (out.), pp. 549–568.

[90] Hopfield, J. J., 1987, "Neural networks and physical systems with emergent collective computational abilities", *Proc. Natl. Acad. Sci. USA*, v. 79, pp. 2554–2558.

[91] Huang, S., 1999, "Gene expression profiling, genetic networks, and cellular states: an integrating concept for tumorigenesis and drug discovery", *J. Mol. Med.*, v. 77, n. 6, pp. 469–480.

[92] Kempe, D., Kleinberg, J., Tardos, E., 2003, "Maximizing the Spread of Influence Through a Social Network". In: *Proc. of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '03, pp. 137–146, New York, NY, USA. ACM.

[93] Kier, L. B., Seybold, P. G., Cheng, C. K., 2005, *Cellular Automata Modeling of Chemical Systems.* Cellular Automata Modeling of Chemical Systems: A Textbook and Laboratory Manual. Springer.

[94] Knapp, E., 1987, "Deadlock detection in distributed databases", *ACM Comput. Surv.*, v. 19, n. 4, pp. 303–328.

[95] Knutson, J., 2011, *A survey of the use of cellular automata and cellular automata-like models for simulating a population of biological cells.* Graduate theses and dissertations, Iowa State University, USA.

[96] Kshemkalyani, A. D., Singhal, M., 1994, “Efficient detection and resolution of generalized distributed deadlocks”, *IEEE T. Software Eng.*, v. 20, n. 1, pp. 43–54.

[97] Kumar, V., Kanal, L. N., 1984, “Parallel Branch-and-Bound Formulations for AND/OR Tree Search”, *IEEE T. on Pattern Anal.*, v. 6, n. 6, pp. 768–778.

[98] Kutten, S., Peleg, D., 1999, “Fault-Local Distributed Mending”, *J. Algorithms*, v. 30, n. 1, pp. 144 – 165.

[99] Laber, E. S., 2008, “A randomized competitive algorithm for evaluating priced AND/OR trees”, *Theor. Comput. Sci.*, v. 401, n. 1–3, pp. 120–130.

[100] Lampis, M., 2012, “Algorithmic meta-theorems for restrictions of treewidth”, *Algorithmica*, v. 64, n. 1, pp. 19–37.

[101] Lewis, K., Gonzalez, M., Kaufman, J., 2012, “Social selection and peer influence in an online social network”, *Proc. Natl. Acad. Sci.*, v. 109, n. 1, pp. 68–72.

[102] Liu, J., Zhou, H., 1994, “Dominating subgraphs in graphs with some forbidden structures”, *Discrete Mathematics*, v. 135, pp. 163–168.

[103] Lovász, L., 1966, “On decomposition of graphs”, *Studia Scientiarum Mathematicarum Hungarica*, v. 1, pp. 237–238.

[104] Ma, X., Dong, B., He, M., 2008, “AND/OR Tree Search Algorithm in Web Service Composition”. In: *Pacific-Asia Workshop on Computational Intelligence and Industrial Application, 2008. PACIIA ’08.*, v. 2, pp. 23–27, Dec.

[105] Mercken, L., Snijders, T. A. B., Steglich, C., et al., 2010, “Dynamics of adolescent friendship networks and smoking behavior”, *Soc. Networks*, v. 32, n. 1, pp. 72–81.

[106] Montanari, A., Saberi, A., 2009, “Convergence to Equilibrium in Local Interaction Games”, *SIGecom Exch.*, v. 8, n. 1 (jul.), pp. 11:1–11:4.

[107] Morabito, R., Pureza, V., 2010, “A heuristic approach based on dynamic programming and and/or-graph search for the constrained two-dimensional guillotine cutting problem”, *Ann. Oper. Res.*, v. 179, n. 1, pp. 297–315.

[108] Moret, B. M. E., 1988, “Planar NAE3SAT is in P”, *SIGACT News*, v. 19, n. 2, pp. 51–54.

[109] Morris, S., 2000, "Contagion", *Rev. Econ. Stud.*, v. 67, pp. 57–78.

[110] Mulzer, W., Rote, G., 2008, "Minimum-weight triangulation is NP-hard", *Journal of the ACM (JACM)*, v. 55, n. 2, pp. 11.

[111] Mustafa, N. H., Pekeč, A., 2004, "Listen to Your Neighbors: How (Not) to Reach a Consensus", *SIAM J. Discret. Math.*, v. 17, n. 4, pp. 634–660.

[112] Nakata, T., Imahayashi, H., Yamashita, M., 2000, "A probabilistic local majority polling game on weighted directed graphs with an application to the distributed agreement problem", *Networks*, v. 35, n. 4, pp. 266–273.

[113] Natanzon, A., Shamir, R., Sharan, R., 2001, "Complexity classification of some edge modification problems", *Discrete Applied Mathematics*, v. 113, n. 1, pp. 109 – 128. Selected Papers: 12th Workshop on Graph-Theoretic Concepts in Computer Science.

[114] Nilsson, N. J., 1971, *Problem-Solving Methods in Artificial Intelligence.* McGraw-Hill Pub. Co.

[115] Oliveira, L. I. L., 2012, *Processos k-Reversíveis em Grafos.* Dissertação de mestrado, Universidade Federal do Rio de Janeiro, Rio de Janeiro, Brazil.

[116] Oliveira, L. I. L., Barbosa, V. C., Protti, F., 2015, "The predecessor-existence problem for k-reversible processes." *Theor. Comput. Sci.*, v. 562, pp. 406–418.

[117] Orponen, P., 1994, "Computational Complexity of Neural Networks: A Survey", *Nordic J. of Computing*, v. 1, n. 1, pp. 94–110.

[118] Papadimitriou, C. H., 1994, *Computational Complexity.* Addison Wesley Pub. Co.

[119] Peleg, D., 1998, "Size bounds for dynamic monopolies", *Discret. Appl. Math.*, v. 86, n. 2–3, pp. 263–273.

[120] Peleg, D., 2002, "Local majorities, coalitions and monopolies in graphs: a review", *Theor. Comput. Sci.*, v. 282, n. 2, pp. 231 – 257. {FUN} with Algorithms.

[121] Perdang, J. M., Lejeune, A., 1993, *Cellular Automata: Prospects in Astrophysical Applications.* World Scientific.

[122] Poljak, S., Sůra, M., 1983, "On periodical behaviour in societies with symmetric influences", *Combinatorica*, v. 3, n. 1, pp. 119–121.

[123] Poljak, S., Turzík, D., 1986, "On an application of convexity to discrete systems", *Discret. Appl. Math.*, v. 13, n. 1, pp. 27 – 32.

[124] Poljak, S., Turzík, D., 1986, "On pre-periods of discrete influence systems", *Discret. Appl. Math.*, v. 13, n. 1, pp. 33 – 39.

[125] Rautenbach, D., 2007, "Dominating and large induced trees in regular graphs", *Discrete Mathematics*, v. 307, pp. 3177–3186.

[126] Rautenbach, D., Volkmann, L., 2007, "New bounds on the k-domination number and the k-tuple domination number", *Applied Mathematics Letters*, v. 20, pp. 98–102.

[127] Ryang, D.-S., Park, K. H., 1995, "A two-level distributed detection algorithm of AND/OR deadlocks", *J. Parallel. Distr. Com.*, v. 28, n. 2, pp. 149–161.

[128] Sahni, S., 1974, "Computationally Related Problems", *SIAM J. Comput.*, v. 3, n. 4, pp. 262–279.

[129] Schaefer, T. J., 1978, "The Complexity of Satisfiability Problems". In: *Proceedings of the Tenth Annual ACM Symposium on Theory of Computing*, STOC '78, pp. 216–226, New York, NY, USA. ACM.

[130] Simon, R., Lee, R. C. T., 1971, "On the Optimal Solutions to AND/OR Series-Parallel Graphs", *J. ACM*, v. 18, n. 3, pp. 354–372.

[131] Singhal, M., 1989, "Deadlock detection in distributed systems", *Computer*, v. 22, n. 11, pp. 37–48.

[132] Snijders, T. A. B., van de Bunt, G. G., Steglich, C. E. G., 2010, "Introduction to stochastic actor-based models for network dynamics", *Soc. Networks*, v. 32, n. 1, pp. 44–60.

[133] Steglich, C., Snijders, T. A. B., Pearson, M., 2010, "Dynamic networks and behavior: separating selection from influence", *Sociol. Methodol.*, v. 40, n. 1, pp. 329–393.

[134] Thorup, M., 1998, "All structured programs have small tree width and good register allocation", *Information and Computation*, v. 142, n. 2, pp. 159–181.

[135] Vazirani, V. V., 2001, *Approximation Algorithms.* New York, NY, USA, Springer-Verlag New York, Inc.

[136] Vichniac, G. Y., 1984, "Simulating physics with cellular automata", *Phys. D: Nonlinear Phenom.*, v. 10, n. 1, pp. 96 – 116.

[137] Šíma, J., Orponen, P., 2003, "General-purpose Computation with Neural Networks: A Survey of Complexity Theoretic Results", *Neural Comput.*, v. 15, n. 12, pp. 2727–2778.

[138] Yannakakis, M., 1981, "Edge-deletion problems", *SIAM Journal on Computing*, v. 10, n. 2, pp. 297–309.

[139] Yannakakis, M., 1978, "Node-and Edge-deletion NP-complete Problems". In: *Proceedings of the Tenth Annual ACM Symposium on Theory of Computing*, STOC '78, pp. 253–264, New York, NY, USA. ACM.

[140] Yarahmadi, Z., Ashrafi, A. R., 2014, "A fast algorithm for computing bipartite edge frustration number of $(3, 6)$-fullerenes", *Journal of Theoretical and Computational Chemistry*, v. 13, n. 02, pp. 1450014.

[141] Zaker, M., 2012, "On dynamic monopolies of graphs with general thresholds", *Discrete Mathematics*, v. 312, pp. 1136–1143.

[142] Zverovich, V., 2015, "On general frameworks and threshold functions for multiple domination", *Discrete Mathematics*, v. 338, pp. 2095–2104.

Appendices

Appendix A:

On f-Reversible Processes on Graphs

This appendix contains the article "*On f-Reversible Processes on Graphs*" presented in *VIII Latin-American Algorithms, Graphs and Optimization Symposium*, 2015, and published in *Electronic Notes in Discrete Mathematics* journal as a proceeding.

Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in DISCRETE MATHEMATICS

Electronic Notes in Discrete Mathematics 50 (2015) 231–236

www.elsevier.com/locate/endm

On f-Reversible Processes on Graphs [2]

Mitre C. Dourado [a,1] Carlos Vinícius G.C. Lima [b,1]
Jayme L. Szwarcfiter [a,b,1]

[a] *Inst. Mat., NCE, Universidade Federal do Rio de Janeiro, Brazil*

[b] *PESC, COPPE, Universidade Federal do Rio de Janeiro, Brazil*

Abstract

Given a graph G and a function $f : V(G) \to \mathbb{N}$ we study the iterative process on G such that, given an initial vertex labelling $c_0 : V(G) \to \{0, 1\}$, each vertex v changes its label if and only if at least $f(v)$ of its neighbors have the different label. It is known that these processes reach periodic behavior after a phase called transient. We give a tight upper bound for transient length and we show that the period is at most two, for all c_0. We also study the problem of finding the minimum number $r_f(G)$ of vertices with initial label 1, such that all vertices reach label 1. Given a constant $k \geq 1$, we show that is NP-complete to determine if $r_f(G) \leq k$, even if G is a bipartite graph with $\Delta(G) \leq 3$ and $f : V(G) \to \{1, 2, 3\}$. We also prove that this problem is NP-complete for planar cubic graphs and $f : V(G) \to \{3\}$, through relationship between $r_f(G)$ and the size of a minimum vertex cover of G, in the case in which $f(v) = d(v)$, for all vertex v.

Keywords: f-reversible processes, maximum transient length, f-conversion set problem, complexity, graph dynamical systems.

[1] *Email addresses:* `mitre@nce.ufrj.br` (Dourado), `gclima@cos.ufrj.br` (Lima), `jayme@nce.ufrj.br` (Szwarcfiter)

[2] Partially supported by CAPES, FAPERJ and CNPq/Brazil.

http://dx.doi.org/10.1016/j.endm.2015.07.039

1 Introduction

Given a finite, undirected and simple graph G, a *process on* G is an infinite sequence $P = (c_t)_{t\in\mathbb{N}} = (c_0, c_1, \ldots)$ of labellings $c_t : V(G) \to \{0, 1\}$ called *configuration* of P and such that $c_t(v)$ denotes the *state* of v, at time $t \in \mathbb{N}$. Moreover, we say that c_t is a *predecessor configuration* of c_{t+1}, for all $t \in \mathbb{N}$, and c_0 is the *initial configuration* of P. This approach is used in a lot of areas. In this work, we consider a special kind of these iterative processes, such that c_{t+1} is obtained from c_t by applying a *threshold function* $f : V(G) \to \mathbb{N}$ to each vertex $v \in V(G)$, so that v changes its state if and only if it has at least $f(v)$ neighbors with the opposite state at t, for each time step $t \in \mathbb{N}$. We say that $f(v)$ is the *threshold value* of v. We call such iterative processes as *f-reversible processes on* G and we denote them by $R_f(G) = (c_t)_{t\in\mathbb{N}}$, given an initial configuration c_0, or simply $R_f(G)$. If f is a k-constant function, then we denote them by $R_k(G)$ and call them *k-reversible processes on* G.

Clearly f-reversible processes are uniquely determined by G, f and c_0 and, due to state updating rule of a vertex v depends only of the states in $N(v)$, the neighborhood of v, we consider only connected graphs. Moreover, denoting the degree of v by $d(v)$, if $f(v) > d(v)$ for some vertex v, then its initial state remains the same for all time. So, we can consider $f(v) = d(v) + 1$. Hence, only threshold functions f satisfying $f : V(G) \to \{0, \ldots, \Delta(G) + 1\}$ will be considered, where $\Delta(G)$ denotes the maximum degree of G. Note that if $f(v) = 0$, then $c_t(v)$ will change for each $t \in \mathbb{N}$.

Because G is finite, c_{t+1} is obtained deterministically from c_t, for all $t \in \mathbb{N}$, and are used only two states, there are 2^n possible configurations and there must exist a time step in which the process becomes periodic, which depends of the initial configuration. The length of the periodic phase is called *period* and it is denoted by $p(c_0) \geq 1$. A *periodic configuration* is one which occurs in the periodic phase. The phase of no periodic configurations is called *transient* and its length is denoted by $\tau(c_0) \geq 0$. Note that the first periodic configuration is obtained at time $\tau(c_0)$. Oliveira, Barbosa and Protti [1] showed that $p(c_0) \leq 2$ and they gave a no tight upper bound for $\tau(c_0)$ of $O(n\Delta(G))$, both for k-reversible processes, $k \geq 2$. Here, we give a sharp upper bound for $\tau(c_0)$, with the same approach of [1], but for general threshold functions. Moreover, we also show that $p(c_0) \leq 2$ in these cases.

We say that an f-reversible process *converges* if all vertices achieve state equal to 1 in $c_{\tau(c_0)}$. So, for convergence we must consider $f(v) > 0$, for all $v \in V(G)$. Given an f-reversible process $R_f(G)$ that converges, the set of vertices with initial state equal to 1 is called an *f-conversion set of* G.

Furthermore, $V(G)$ is a trivial f-conversion set. We also study the problem of finding the minimum cardinality of an f-conversion set of G, denoted by $r_f(G)$. Similarly, we denote $r_k(G)$ if f is constant with value $k \geq 1$. Dreyer [2] proved that finding $r_k(G)$ is NP-hard for $k \geq 3$ and gave exact values for some specific graph classes. Dourado et al. [3] showed the NP-hardness for $k = 2$. For $k = 1$, the trivial solution is the unique. We show that determining if $r_f(G) \leq k$ is NP-complete even if G is bipartite with $\Delta(G) \leq 3$ and $f : V(G) \to \{1, 2, 3\}$. Furthermore, we also show the NP-completeness for 3-reversible processes on planar cubic graphs, by relationship between the size of a minimum *vertex cover* of G and $r_f(G)$, in the cases in which $f(v) = d(v)$, for all $v \in V(G)$.

2 Period and Transient Length

We show that $p(c_0) \leq 2$, whose upper bound occurs for example in $R_1(K_2)$, whose vertices have opposite states. To this end, we prove that f-reversible processes are particular cases of *threshold networks*, denoted by (A, b), which are defined by a quadratic matrix A and a *threshold vector* b. The vertex states of threshold networks belong also to $\{0, 1\}$, for example, and configuration c_{t+1} is given from c_t, defining an iterative process, according to the following rule: $c_{t+1}(v_i) = 1$, if and only if $\sum_{j=1}^{n} A_{ij} c_t(v_j) - b_i \geqslant 0$. It was established in [4] that $p(c_0) \leq 2$ if A is symmetric. Hence, our approach consists in finding appropriate symmetric matrix and threshold vector. We do it for f_1-f_2-reversible processes, in which there are two threshold functions f_1 and f_2, where v changes its state from 0 to 1 according f_1 and from 1 to 0 according f_2. If $f_1 = f_2$, then we obtain an f-reversible process, with $f_1 = f_2 = f$.

Theorem 2.1 *f_1-f_2-Reversible processes on G are particular threshold networks (A, f_1), whose matrix A is given as the adjacency matrix of G, such that $A_{ii} = f_1(v_i) + f_2(v_i) - d(v_i) - 1$, for all $1 \leq i \leq n$.*

In the proof of Theorem 2.1, we show that $c_{t+1}(v) \neq c_t(v)$ if and only if v has at least $f(v)$ neighbors with different state at t. We consider the switch cases from 0 to 1 and 1 to 0 according to update rule of threshold networks.

Corollary 2.2 *The period of f_1-f_2-reversible processes is at most two.*

Now, we will present a polynomial tight upper bound for transient length of f-reversible processes on G. For all $t \in \mathbb{N}$, let $S_1(t)$ and $S_2(t)$ be a partition of $V(G)$, such that $S_1(t)$ denotes the vertices that change their states at instant t and $S_2(t)$ those that do not. Given a vertex v, we denote the number of neighbors with the opposite state of v at time t by $op_t(v)$. So, we can denote the previous sets by $S_1(t) = \{v : op_t(v) \geq f(v)\}$ and $S_2(t) = \{v : op_t(v) < f(v)\}$. We modify slightly the approach presented by Oliveira, Barbosa and Protti [1] for k-reversible processes, so that it works to general threshold functions f. The *energy function* of an f-reversible process at time t is given as:

$$E(t) = \sum_{v \in S_1(t)} (op_t(v) - f(v)) + \sum_{v \in S_2(t)} (f(v) - op_t(v))$$

Theorem 2.3 gives an upper bound for transient length based in the energy value and cardinality of S_2 at time $\tau(c_0) - 1$, last time of transient phase.

Theorem 2.3 *If $\tau(c_0) > 0$, then $\tau(c_0) \leq E(\tau(c_0) - 1) - |S_2(\tau(c_0) - 1)| + 1$. Furthermore, the bound is attained.*

The idea of the proof is to first show that the energy function never decrease. Hence, it reach a maximum value, denoted by E_{max}, because it can not increase in the periodic phase. We analyse the energy variation of each vertex v from time t to $t + 1$ in all cases in which v change between S_1 and S_2, or otherwise. So, we relate $\tau(c_0)$ with the energy variation from time 0 to $\tau(c_0) - 1$. Corollary 2.4 gives the tight upper bound based on graph size, denoted by n, and the maximum and minimum threshold values, f_{max} and f_{min}, respectively. Note that it is restricted to processes with period equal to one. We define the following sets used in its proof:

- $X_1 = \{v \in V(G) | c_{\tau(c_0)-1}(v) = 0 \text{ and } v \in S_1(\tau(c_0) - 1)\}$
- $Y_1 = \{v \in V(G) | c_{\tau(c_0)-1}(v) = 1 \text{ and } v \in S_1(\tau(c_0) - 1)\}$

Corollary 2.4 *If $\tau(c_0) > 0$ and $p(c_0) = 1$, then, for all c_0:*

$$\tau(c_0) \leq \begin{cases} (n-1)(f_{max} - 1) - (f_{min} - 1) & , \textit{ if } X_1 \textit{ or } Y_1 \textit{ is empty;} \\ (n-1)(f_{max} - 1) - f_{max} & , \textit{ otherwise.} \end{cases}$$

Figure 1 shows an example illustrating that Theorem 2.3 and Corollary 2.4 are sharp. The gray vertices have state one and the number above vertex v represents $f(v)$. We consider odd cycles $C_n = v_1 v_2 \ldots v_n$, so that $f(v_1) = 1$ and $f(v_i) = 2$, for all $i \neq 1$. Furthermore, $c_0(v_1) = 1$ and $c_0(v_i) = 1$ if and only if i is even and $i > 0$. Actually, the bound of Corollary 2.4 can be arbitrarily far from $\tau(c_0)$. For example, consider an f-reversible process on a star G with n vertices and central vertex v, such that $f(v) = n$ and $f(u) = 1$, for all $u \neq v$. If $c_0(v) = 1$ and $c_0(u) = 0$, for all $u \neq v$, then $\tau(c_0) = 1$. But, Corollary 2.4, give $\tau(c_0) \leq (n-1)(f_{max} - 1) - (f_{min} - 1) = (n-1)^2$. However, Theorem 2.3 give us the correct value in this case. In this way, a characterization of this limit in terms of G, c_0 and f remains an interesting problem.

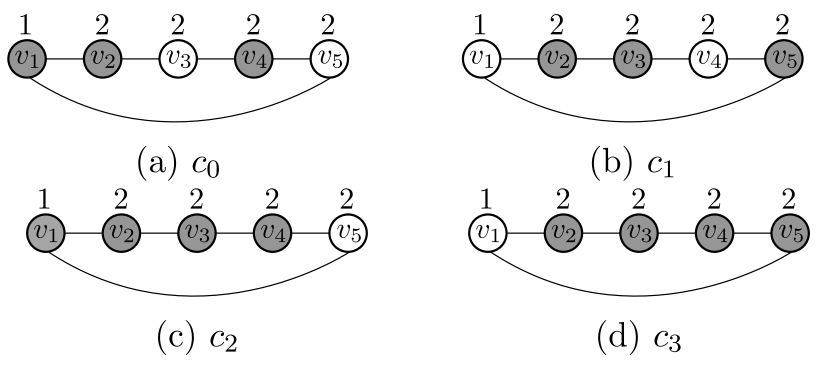


Fig. 1. Transient configurations of $R_f(C_5)$.

3 NP-Completeness of f-Conversion Set Problem

In this section, we prove the NP-completeness of f-Conversion Set Problem. The proof is a reduction through a restriction of 3SAT, where each clause has 2 or 3 literals and each variable occurs in at most 3 clauses [5]. It can be polynomially solved if all clauses have exactly 2 or exactly 3 literals [6].
f-Conversion Set Problem
Instance: A graph G, a function $f : V(G) \to \mathbb{N}$ and an integer $k \geqslant 1$.
Question: Is $r_f(G) \leqslant k$?

Determining whether $r_2(G) \leq k$ was proved to be NP-hard by Dourado et al. [3]. Denoting the neighborhood of a vertex v in a vertex set X by $N_X(v)$, they also determined that if there exist two disjoint vertex subsets A and B where each vertex of A has different state of the vertices of B, $|N_B(v)| \geq f(v)$, for all $v \in A$ and $|N_A(u)| \geq f(u)$, for all $u \in B$, then the process does not converge. Based in this observation, we give a simple lemma:

Lemma 3.1 *Let $R_f(G)$ be an f-reversible process on G which converges. If there exists a vertex $v \in V(G)$ such that $f(v) = d(v)$ and $f(u) = 1$, for all $u \in N(v)$, then $c_t(v) = 1$ and $c_t(u) = 1$, for some $u \in N(v)$ and for all $t \geq 0$.*

Theorem 3.2 *f-Conversion Set Problem is NP-complete, even if G is a bipartite graph with maximum degree 3 and $f : V(G) \to \{1, 2, 3\}$.*

Sketch of proof: The problem is in NP, since the process has a polynomial number of steps, by Corollary 2.4, and each one is executed in $O(n+m)$. Moreover, given a restricted 3SAT problem formula F with variables $X_1, \dots, X_n$ and clauses $C_1, \dots, C_m$, we construct a graph G, such that: for each X_i, G contains three vertices x_i, $\overline{x_i}$ and a_i; for each clause C_j, we assign one vertex c_j; we add edges $a_i x_i$ and $a_i \overline{x_i}$ for all $1 \leq i \leq n$, and $x_i c_j$ if and only if $X_i \in C_j$; finally we assign $f(x_i) = f(\overline{x_i}) = 1$ and $f(v) = d(v)$, to the other vertices. By Lemma 3.1 we prove that F is satisfying if and only if $r_f(G) = m + 2n$. $\square$

We also analyse the complexity of finding a minimum f-conversion set on general graphs, such that $f(v) = d(v)$ for all $v \in V(G)$. In this case, if an edge is such that the states of its ends are the same, then these vertices never will change their states, showing that $r_f(G) \geq \beta(G)$, the size of a minimum vertex cover[7] of G. We stablish the following theorem relating $r_f(G)$ and $\beta(G)$.

Theorem 3.3 *Given an f-reversible process on a connected graph G, such that $f(v) = d(v)$ for all $v \in V(G)$, if exists a minimum vertex cover of G which is not an independent set, then $r_f(G) = \beta(G)$. Otherwise, $r_f(G) = \beta(G) + 1$.*

Until this moment, approximation results on f-Conversion Set Problem are unknown. By König-Egervry Theorem [7], if G is a bipartite graph, then $\beta(G)$ can be found in polynomial time. Thus, by second case of Theorem 3.3 we can determine $r_f(G)$ in polynomial time for bipartite graphs. Moreover, because Vertex Cover Problem [5] is NP-complete even for planar cubic graphs, follows that:

Corollary 3.4 f-Conversion Set Problem *is* NP*-complete even if G is a planar cubic graph and $f : V(G) \to \{3\}$. But we can determine $r_f(G)$ in polynomial time if G is connected bipartite and $f(v) = d(v)$, for all $v \in V(G)$.*

References

[1] Oliveira, L. I. L., V. C. Barbosa and F. Protti, *An energy function and its application to the periodic behavior of k-reversible processes.* (2013).

[2] Dreyer, P. A., *Application and Variations of Domination in Graphs,* Ph.d. dissertation, Rutgers University (2000).

[3] Dourado, M., L. Penso, D. Rautenbach and J. Szwarcfiter, *Reversible iterative graph processes.*, Theor. Comput. Sci. **460** (2012), pp. 16–25.

[4] Goles-Chacc, E., F. Fogelman-Soulie and D. Pellegrin, *Decreasing energy functions as a tool for studying threshold networks*, Discrete Appl. Math. **12** (1985), pp. 261–277.

[5] Garey, M. and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-completeness.* W. H. Freeman and Co., 1979.

[6] Papadimitriou, C., *Computational Complexity.* Addison Wesley, 1994.

[7] Bondy, A. and U. Murty., *Graph Theory.* Spring-Verlag Press, 2008.

Appendix B:

A Computational Study of *f*-Reversible Processes on Graphs

This appendix contains the article "*A Computational Study of f-Reversible Processes on Graphs*" submitted on December 2015 to *Discrete Applied Mathematics* journal.

A Computational Study of f-Reversible Processes on Graphs

Carlos V.G.C. Lima[c,*], Leonardo I.L. Oliveira[c], Valmir C. Barbosa[c], Mitre C. Dourado[b], Fábio Protti[a], Jayme L. Szwarcfiter[b,c]

[a]*Instituto de Computação, Universidade Federal Fluminense, Rua Passo da Pátria, 156, 24210-240 Niterói – RJ, Brazil.*
[b]*Instituto de Matemática, NCE, and COPPE, Universidade Federal do Rio de Janeiro, Rio de Janeiro – RJ, Brazil.*
[c]*Programa de Engenharia de Sistemas e Computação, COPPE, Universidade Federal do Rio de Janeiro, Caixa Postal 68511, 21941-972 Rio de Janeiro – RJ, Brazil.*

Abstract

An f-reversible process on a graph $G = (V, E)$ is a graph dynamical system on $V(G)$ defined as follows. Given a function $f : V(G) \rightarrow \mathbb{N}$ and an initial vertex labelling $c_0 : V(G) \rightarrow \{0, 1\}$, every vertex v changes its label if and only if at least $f(v)$ of its neighbors have the opposite state, synchronously in discrete-time. For such processes, we present a new nondecreasing time function similar to the monotonically decreasing energy functions used to study threshold networks, which leads to a periodic behavior after a transient phase. Using this new function, we provide a tight upper bound on the transient length of f-reversible processes. Furthermore, we prove that it is equal to $n - 3$ for trees with $n \geq 4$ vertices and $\text{Im}_f = \{2\}$. Moreover we present an algorithm that generates all the initial configurations attaining this bound and we prove that the number of such configurations is $O(n)$. We also consider the problem of determining the smallest number $r_f(G)$ of vertices with initial label 1 for which all the vertices eventually reach label 1 after the complete evolution of the dynamics, which models consensus problems on networks. We prove that it is NP-hard to compute $r_f(G)$ for bipartite graphs with $\Delta(G) \leq 3$ and $\text{Im}_f = \{1, 2, 3\}$. Finally, we prove that $\beta(G) \leq r_f(G) \leq \beta(G) + 1$, when $f(v) = d(v)$ for all $v \in V(G)$, where $\beta(G)$ is the size of a minimum vertex cover of G.

Keywords: f-Reversible processes, Maximum transient length, Minimum f-conversion set, Graph dynamical systems, Energy functions, Consensus

*Corresponding author
Email addresses: `gclima@cos.ufrj.br` (Carlos V.G.C. Lima), `leonardo@cos.ufrj.br` (Leonardo I.L. Oliveira), `valmir@cos.ufrj.br` (Valmir C. Barbosa), `mitre@dcc.ufrj.br` (Mitre C. Dourado), `fabio@ic.uff.br` (Fábio Protti), `jayme@nce.ufrj.br` (Jayme L. Szwarcfiter)

1. Introduction

Let $G = (V, E)$ be a simple, undirected, finite graph with n vertices and m edges. A *discrete dynamical process on* G is an infinite sequence $\mathcal{P} = (c_t)_{t\in\mathbb{N}} = (c_0, c_1, \dots)$ of *configurations* $c_t : V(G) \to \{0, 1\}$. We say that c_0 is the *initial configuration* of $\mathcal{P}$ and $c_t(v)$ denotes the *state* of v at time $t \in \mathbb{N}$. The transition from c_t to c_{t+1} takes place under the same predetermined rule at each time step. This approach is employed on a wide variety of areas such as social influence [9, 17, 28, 41–43], gene expression networks [27], immune systems [2], cellular automata [3], statistical mechanics [1, 4], marketing strategies [11, 13, 28], finite discrete dynamical systems [5, 37, 45], astrophysics and physics [20, 40, 44], opinion and disease [10, 32] dissemination, simulations of biologic cell populations [30], modeling of chemical systems [29], neural networks [22], local interaction games [33, 34] and distributed computing [15, 16, 25, 31, 35, 39].

In this work we consider a special kind of iterative processes on simple undirected graphs that generalizes the majority voting approach studied by Peleg [39]. Formally, the update rule is such that c_{t+1} is obtained from c_t according to a *threshold function* $f : V(G) \to \mathbb{N}$ applied to each vertex $v \in V(G)$ and at each time $t \in \mathbb{N}$. Denoting the *neighborhood* of v by $N(v)$, the update rule is defined as

$$c_{t+1}(v) = \begin{cases} 1 - c_t(v) & \text{, if } |\{u \in N(v) : c_t(u) \neq c_t(v)\}| \geqslant f(v); \\ c_t(v) & \text{, otherwise.} \end{cases} \tag{1}$$

In other words, every vertex v changes its state if and only if it has at least $f(v)$ neighbors with the opposite state, at each time step $t \in \mathbb{N}$. Moreover, the state updates are done synchronously.

We say that $f(v)$ is the *threshold value* of v, which does not change during the process. Given an initial configuration c_0, we call such an iterative process following Eq. (1) as an *f-reversible process on* G, denoted by $R_f(G, c_0) = (c_t)_{t\in\mathbb{N}}$, or simply $R_f(G, c_0)$. If f is a k-constant function, it is denoted by $R_k(G, c_0)$ and called *k-reversible process on* G, $k \in \mathbb{N}$.

An f-reversible process is analogous to the majority voting approach studied by Mustafa and Pekeč [35] with $f(v) = \left\lfloor \frac{d(v)}{2} \right\rfloor + 1$, where $d(v)$ denotes the *degree* of v. In fact, several works consider iterative processes on graphs where changes of states are allowed at most once during the process [7, 8, 13, 15]. These approaches are used, for instance, in contexts of opinion spread on a social network or disease dissemination among individuals. Gargano *et al.* [19] considered the influence diffusion in social networks in which some individuals are “actives” to influence their neighbors for a limited number of time steps, once they were influenced by a sufficient number of neighbors. However, if a vertex has been influenced then it remains in this state forever. An f-reversible process represents the extremal case in which the elements do not have “memory”. Thus, the vertices do not keep themselves influenced if they have the required amount of neighbors to change their states and each one needs to be convinced by a subset of neighbors to change its opinion, at each time step.

We deal with two problems regarding f-reversible processes. The first one concerns on its periodic behavior, where we present the following results:

1. We show that f-reversible processes are particular cases of *threshold networks*, which have at most two configurations in their periodic phase. This result is presented in Theorem 1;
2. Using a so called *energy function*, Goles *et al.* [24] provided an upper bound on the number of time steps needed to achieve the periodic behavior (maximum transient length). In this work we present a more intuitive function which provides a tight upper bound based on G, f and c_0. This result is given in Theorem 6;
3. We also analyze the maximum transient length of 2-reversible processes on trees with at least 5 vertices, where we prove that it is at most $n-3$ in Theorem 7. The proof yields an algorithm (Algorithm 1) that generates all initial configurations that allow the process to reach $n-3$ time steps. Moreover, we prove that the number of such configurations is $O(n)$.

The second problem studied refers to finding the minimum cardinality of a vertex subset which allows that all the vertices of G reach same state 1. This problem was proved to be NP-hard even for 2-reversible processes on general graphs [12]. In the same work, this problem was left open for paths and cycles, where $\mathrm{Im}_f = \{1, 2\}$. Dreyer [14] determined in his thesis the exact value for some specific graph classes, where f is constant. Moreover he proved that such a problem is NP-complete for k-regular graphs, where the function is k-constant. Regarding this problem, we obtain the following results:

1. We prove that it is NP-hard to determine such a parameter for bipartite graphs whose maximum degree is at most 3 and $\mathrm{Im}_f = \{1, 2, 3\}$. This result is presented in Theorem 11;
2. Denoting the size of a minimum *vertex cover* of G by $\beta(G)$, we prove that such a parameter is equal to $\beta(G)$ or $\beta(G) + 1$, when $f(v) = d(v)$ for all $v \in V(G)$. Theorem 12 presents this result. As a corollary, if G is not a bipartite graph then $\beta(G)$ is the exact number of vertices required.

2. Definitions, Properties, and Main Results

We refer to [6] for definitions and concepts related to the theory of graphs. It is clear that an f-reversible process is uniquely determined by G, f and c_0. Furthermore, due to Eq. (1), every vertex v depends on only its own state and on the states of its neighbors to define its next one. Thus, we will consider only connected graphs in the remaining of the paper. Moreover, if $f(v) > d(v)$ for some vertex v then its initial state does not change during the whole process. Therefore we can assume $f(v) = d(v) + 1$ in this case. Hence, only threshold functions f satisfying $\mathrm{Im}_f = \{0, \ldots, \Delta(G) + 1\}$ will be considered, where $\Delta(G)$ denotes the maximum degree of G. Note that if $f(v) = 0$ then the state of v changes at every time step $t \in \mathbb{N}$.

2.1. The Periodic Behavior

Let $R_f(G, c_0)$ be an f-reversible process. Since G is finite and c_{t+1} is obtained deterministically from c_t, according to Eq. (1), the number of possible configurations is equal to 2^n. Hence, there must exist a finite time step in which the process becomes periodic. The set of configurations preceding the periodic phase is called *transient* and its length is denoted by $\tau(c_0) \geq 0$. The length of the periodic phase is called *period*, denoted by $p(c_0) \geq 1$. A *periodic configuration* is one that occurs in the periodic phase and note that the first one is reached at time $\tau(c_0)$. Formally, the period and transient lengths satisfy the following conditions:

- $c_{t+p(c_0)} = c_t$, for all $t \geq \tau(c_0)$;
- $c_{t+q} \neq c_t$, for all ($t < \tau(c_0)$ and $q \geq 1$) or ($t \geq \tau(c_0)$ and $1 \leq q < p(c_0)$).

Two natural parameters arise from the above definitions. Given a threshold function f and a graph G, denote the largest transient length over all initial configurations by $\tau_f(G)$ (resp. $\tau_k(G)$ if f is a k-constant function). Analogously, let $p_f(G)$ be (resp. $p_k(G)$ if f is a k-constant function) the largest period over all initial configurations.

Oliveira, Barbosa and Protti [36] studied the problem of determining if a configuration c has a predecessor configuration, which is one that reaches c within exactly one time step. They have dealt only with k-reversible processes, where they proved that determining if there exits a predecessor configuration of one given it is NP-complete for bipartite graphs, but linear time solvable for trees. Moreover they also present a linear time algorithm for 2-reversible processes on graphs whose maximum degree is at most 3. They also dealt with the problem of counting the number of predecessor configurations, in which they showed an $O(n^2)$ time algorithm for trees.

Dreyer [14] proved that $\tau_k(G)$ is $O(m + n^2)$ and $p_k(G) \leq 2$, where these results are based on reductions from the so-called *threshold networks* [22, 23]. It is known that the maximum period of threshold networks is at most 2 [24, 41]. An intuitive approach to prove this result is based on a monotonic function called *energy function*. Its definition is very similar to that of the energy function associated with Hopfield networks [26]. It is a Lyapunov function and can be used to prove several results associated with the period and transient lengths of threshold and majority networks. This function dates back to Ref. [24] (page 269, inside the proof of Proposition 2) and was later reproduced in Ref. [21] (page 70, Eq. (3.3)).

2.2. The Potential Function

Let us consider an f-reversible process on a graph G. For all $t \in \mathbb{N}$, let $S_1(t)$ and $S_2(t)$ be a bipartition of $V(G)$, where $S_1(t)$ denotes the set of vertices which change their states at time t and $S_2(t)$ the set those that do not. Given a vertex v, we denote the number of neighbors with the opposite state of v at t by $op_t(v)$. Thus, it follows that

$$S_1(t) = \{v : op_t(v) \geq f(v)\}$$

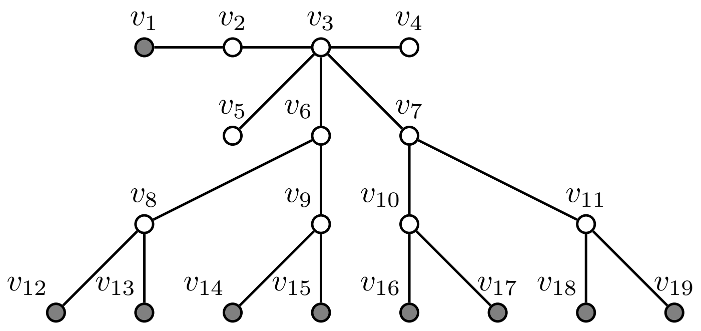


Figure 1: The initial configuration of a 2-reversible process on a tree whose gray vertices have state 1.

and

$$S_2(t) = \{v : op_t(v) < f(v)\}.$$

We define a nonnegative function that we call *potential function* $P(t)$ for f-reversible processes as

$$P(t) = \sum_{v \in S_1(t)} (op_t(v) - f(v)) + \sum_{v \in S_2(t)} (f(v) - op_t(v)) \,. \tag{2}$$

Fig. 2 contains plots of the potential and energy functions against time. One of them (filled line) is the energy function of Ref. [24] (page 269, inside the proof of Proposition 2), later reproduced in Ref. [21] (page 70, Eq. (3.3)). The other (dotted line) refers to the potential function of Eq. (2). Data in the plots correspond to the tree depicted in Fig. 1. Both plots refer to a 2-reversible process and, in the case of the energy function of Ref. [24], to $b_i = 1.5$ as the additional required parameter for every vertex v_i. The initial configuration has 0 at vertices v_2–v_{11} and 1 at all others. The two functions differ markedly and no simple reduction seems to exist to transform one into the other. In particular, the potential function is nondecreasing (rather than monotonically decreasing), as illustrated by Fig. 2.

2.3. The Minimum f-Conversion Set Problem

We say that an f-reversible process $R_f(G, c_0)$ is *uplifting* if all vertices achieve state 1 in a finite number of time steps, which is given by $\tau(c_0)$. An f-*conversion set* of G is a subset of vertices whose initial states are equal to 1 and for which the process is uplifting. For this end, we cannot consider any threshold values equal to 0. Furthermore $V(G)$ is a trivial f-conversion set of G. We also study the problem of finding the minimum cardinality of an f-conversion set of G, denoted by $r_f(G)$ ($r_k(G)$ if f is a k-constant function, $k \geq 1$). For $k = 1$, the trivial solution given by $V(G)$ is unique. We consider the following decision problem in this work:

***f*-Conversion Set**

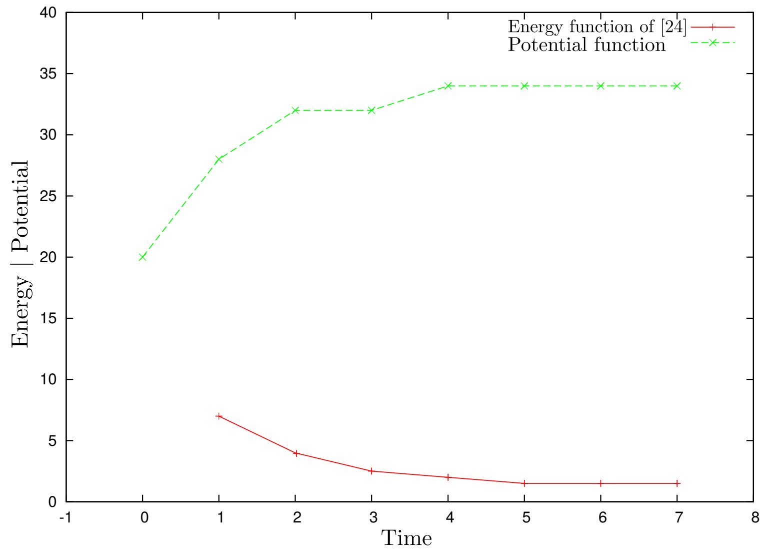


Figure 2: Time evolution of the potential (dotted line) and energy (filled line) functions.

Input: A graph G, a function $f : V(G) \to \mathbb{N}$ and an integer $q \geq 1$.
Question: Is $r_f(G) \leq q$?

In his thesis, Dreyer [14] proved that f-Conversion Set is NP-complete for k-reversible processes on k-regular graphs, $k \geq 3$. Moreover he gave exact values of $r_k(G)$ for some specific graph classes. Dourado *et al.* [12] showed the NP-hardness of determining $r_2(G)$ for general graphs G. They also stated an algorithm based on dynamic programming which computes $r_f(P_n)$ for specific paths P_n with n vertices. They consider both threshold values 1 and 2, where every v with $f(v) = 1$ has a neighbor u with $f(u) = 1$. Thus, computing $r_f(G)$ even for paths and cycles remains open.

The remainder of the text is organized as follows. In Section 3 we prove that f-reversible processes are particular cases of threshold networks whose matrix is symmetric, proving that $p_f(G) \leq 2$. We also present a sharp upper bound for $\tau_f(G)$, based on the potential function given by Eq. (2). To this end we prove that such a function is monotonically non-decreasing. In Section 4 we show that $\tau_2(T) \leq n - 3$, for trees T with $n \geq 4$. Moreover, we present all the initial configurations reaching such a bound. We also present an algorithm that generate all the initial configurations achieving $n - 3$ time steps until the uplifting is reached and the number of such configurations. In Section 5 we prove that f-Conversion Set is NP-complete for bipartite graphs G with $\Delta(G) \leq 3$ and $\text{Im}_f = \{1, 2, 3\}$. It also contains the proof that $r_f(G) \in \{\beta(G), \beta(G) + 1\}$, when $f(v) = d(v)$ for all $v \in V(G)$. Section 6 contains our conclusions.

3. The Period and the Transient Length

In this section we prove that the potential function given by Eq. (2) is nondecreasing. Thus, we show how it can be used to give a tight upper bound for the maximum transient length. Next we consider the maximum transient length of 2-reversible processes on trees. But first, we concern ourselves with the maximum period of f-reversible processes.

3.1. The Maximum Period

We prove that $p_f(G) \leq 2$, where this bound occurs for example in an 1-reversible process on K_2 whose vertices have opposite states. To this end, we prove that f-reversible processes are particular cases of threshold networks (A, b), which are defined by a square matrix A and a *threshold vector* b. The vertex states also belong to $\{0, 1\}$, for example, and configuration c_{t+1} is obtained from c_t, defining an iterative process such that: $c_{t+1}(v_i) = 1$ if and only if $\sum_{j=1}^{n} A_{ij} c_t(v_j) - b_i \geq 0$. It was established in [24] that $p_f(G) \leq 2$ if A is symmetric. Hence, our approach consists on finding appropriate symmetric matrix and threshold vector. In fact, we do it for (f_1,f_2)-*reversible processes*, in which there are two threshold functions f_1 and f_2, where v changes its state from 0 to 1 according to f_1 and from 1 to 0 according to f_2. If $f_1 = f_2$ then we obtain an f-reversible process, with $f = f_1 = f_2$.

Theorem 1. *An (f_1,f_2)-reversible process on G is a threshold network (A, f_1) for which*

$$A_{ij} = \begin{cases} 1 & \text{, if } v_iv_j \in E(G) \text{ and } i \neq j; \\ 0 & \text{, if } v_iv_j \notin E(G) \text{ and } i \neq j; \\ f_1(v_i) + f_2(v_i) - d(v_i) - 1 & \text{, if } i = j. \end{cases}$$

Proof. Let $n_t^1(v_i)$ and $n_t^0(v_i)$ be the number of neighbors of v_i with states 1 and 0 at time t, respectively. It is enough to show that $c_{t+1}(v_i) \neq c_t(v_i)$ if and only if v_i has at least $f_1(v_i)$ neighbors with state 1 or v_i has at least $f_2(v_i)$ neighbors with state 0 at time t, if v has states 0 or 1, respectively.

• $c_t(v_i) = 1$: we get that $c_{t+1}(v_i) = 0$ if and only if $\sum_{j=1}^{n} A_{ij} c_t(v_j) - b_i < 0$. Hence:

$$\begin{aligned} \sum_{j=1}^{n} A_{ij} c_t(v_j) &< f_1(v_i); \\ (f_1(v_i) + f_2(v_i) - d(v_i) - 1)c_t(v_i) + n_t^1(v_i) + n_t^0(v_i) &< f_1(v_i); \\ f_1(v_i) + f_2(v_i) - d(v_i) - 1 + n_t^1(v_i) &< f_1(v_i); \\ f_2(v_i) - (n_t^1(v_i) + n_t^0(v_i)) - 1 + n_t^1(v_i) &< 0; \\ f_2(v_i) - n_t^0(v_i) - 1 &< 0; \\ f_2(v_i) &< n_t^0(v_i) + 1; \\ f_2(v_i) &\leqslant n_t^0(v_i). \end{aligned}$$

In this way, v_i changes its state from 1 to 0 if and only if $f_2(v_i) \leq n_t^0(v_i)$, as (f_1,f_2)-reversible processes.

• $c_t(v_i) = 0$: we get that $c_{t+1}(v_i) = 1$ if and only if $\sum_{j=1}^{n} A_{ij} c_t(v_j) - b_i \geqslant 0$. Hence:

$$\begin{aligned} \sum_{j=1}^{n} A_{ij} c_t(v_j) &\geqslant f_1(v_i); \\ (f_1(v_i) + f_2(v_i) - d(v_i) - 1) c_t(v_i) + n_t^1(v_i) + n_t^0(v_i) &\geqslant f_1(v_i); \\ n_t^1(v_i) &\geqslant f_1(v_i). \end{aligned}$$

Analogously, v_i changes its state from 0 to 1 if and only if $f_1(v_i) \leq n_t^1(v_i)$, as (f_1,f_2)-reversible processes. □

Corollary 2. *The period of (f_1,f_2)-reversible processes is at most 2.*

Proof. Since A is symmetric, the corollary follows directly from the result of Ref. [24]. □

3.2. Monotonicity of the Potential Function

First, we will prove that the potential function given by Eq. (2) is equivalent to

$$P'(t) = \sum_{v \in S_1(t)} (op_{t+1}(v) - f(v)) + \sum_{v \in S_2(t)} (f(v) - op_{t+1}(v)). \tag{3}$$

To this end, we consider a partition of the edges whose ends have opposite states at time t as follows:

- $A(t) = \{(u,v) \in E(G) :\ u \in S_1(t),\ v \in S_1(t) \text{ and } c_t(u) \neq c_t(v)\}$;
- $B(t) = \{(u,v) \in E(G) :\ u \in S_2(t),\ v \in S_2(t) \text{ and } c_t(u) \neq c_t(v)\}$;
- $C(t) = \{(u,v) \in E(G) \setminus \{A(t) \cup B(t)\} \text{ and } c_t(u) \neq c_t(v)\}$.

Note that each edge of $C(t)$ does not have both ends in the same set $S_1(t)$ or $S_2(t)$. Hence, we get that

$$\sum_{v \in S_1(t)} op_t(v) = 2|A(t)| + |C(t)| \quad \text{and} \quad \sum_{v \in S_2(t)} op_t(v) = 2|B(t)| + |C(t)|.$$

Therefore,

$$\sum_{v \in S_1(t)} op_t(v) - \sum_{v \in S_2(t)} op_t(v) = 2(|A(t)| - |B(t)|). \tag{4}$$

Lemma 3. *For $t \geq 0$, $P(t) = P'(t)$.*

Proof. Observe that $P(t)$ and $P'(t)$ can be rewritten as

$$P(t) = \left(\sum_{v \in S_1(t)} op_t(v) - \sum_{v \in S_2(t)} op_t(v) \right) + \left(\sum_{v \in S_2(t)} f(v) - \sum_{v \in S_1(t)} f(v) \right),$$

$$P'(t) = \left(\sum_{v \in S_1(t)} op_{t+1}(v) - \sum_{v \in S_2(t)} op_{t+1}(v) \right) + \left(\sum_{v \in S_2(t)} f(v) - \sum_{v \in S_1(t)} f(v) \right).$$

Thus, it is enough to prove that

$$\sum_{v \in S_1(t)} op_t(v) - \sum_{v \in S_2(t)} op_t(v) = \sum_{v \in S_1(t)} op_{t+1}(v) - \sum_{v \in S_2(t)} op_{t+1}(v). \tag{5}$$

We can observe that both terms of Eq. (5) are defined on the same sets, but referring to the number of neighbors with opposite states at sequential instants, to each vertex v. We also consider similar sets to $A(t)$, $B(t)$ and $C(t)$, but referring to edges whose ends have opposite states at time $t+1$:

- $A'(t) = \{(u,v) \in E(G): \ u \in S_1(t), \ v \in S_1(t)$ and $c_{t+1}(u) \neq c_{t+1}(v)\}$;
- $B'(t) = \{(u,v) \in E(G): \ u \in S_2(t), \ v \in S_2(t)$ and $c_{t+1}(u) \neq c_{t+1}(v)\}$;
- $C'(t) = \{(u,v) \in E(G) \setminus \{A'(t) \cup B'(t)\}$ and $c_{t+1}(u) \neq c_{t+1}(v)\}$.

Since all vertices in $S_1(t)$ change their states and all vertices in $S_2(t)$ do not, all edges of $A(t)$ appear in $A'(t)$. The same holds for all edges of $B(t)$ in $B'(t)$. Moreover, since $A(t)$ and $A'(t)$ are defined over edges whose ends are in same set $S_1(t)$, we get that $A(t) = A'(t)$. It is analogous for $B(t)$ and $B'(t)$ with respect to $S_2(t)$. Therefore,

$$\sum_{v \in S_1(t)} op_{t+1}(v) - \sum_{v \in S_2(t)} op_{t+1}(v) = 2(|A'(t)| - |B'(t)|). \tag{6}$$

Eq. (4) and Eq. (6) show that Eq. (5) is true, completing the proof. □

Let $\Delta P(t)$ be the variation of the potential function from t to $t+1$, i.e., $\Delta P(t) = P(t+1) - P(t)$. Now, we will show that the potential variation is not negative, for all $t \in \mathbb{N}$.

Lemma 4. *$P(t)$ is a nondecreasing function.*

Proof. By Lemma 3, we can rewrite the potential variation as $\Delta P(t) = P(t+1) - P'(t)$. Therefore,

$$\begin{aligned}\Delta P(t) = &\sum_{v \in S_1(t+1)} (op_{t+1}(v) - f(v)) + \sum_{v \in S_2(t+1)} (f(v) - op_{t+1}(v)) \\ &- \sum_{v \in S_1(t)} (op_{t+1}(v) - f(v)) - \sum_{v \in S_2(t)} (f(v) - op_{t+1}(v)).\end{aligned}$$

Now, we can observe the contribution of each vertex to $\Delta P(t)$:

- If $v \in S_1(t)$ and $v \in S_1(t+1)$: $op_{t+1}(v) - f(v) - op_{t+1}(v) + f(v) = 0$;
- If $v \in S_2(t)$ and $v \in S_2(t+1)$: $f(v) - op_{t+1}(v) - f(v) + op_{t+1}(v) = 0$;
- If $v \in S_1(t)$ and $v \in S_2(t+1)$:

  - $f(v) - op_{t+1}(v) - op_{t+1}(v) + f(v) = 2(f(v) - op_{t+1}(v)) > 0$, since $f(v) > op_{t+1}(v)$;

- If $v \in S_2(t)$ and $v \in S_1(t+1)$:
  - $op_{t+1}(v) - f(v) - f(v) + op_{t+1}(v) = 2(op_{t+1}(v) - f(v)) \geq 0$, since $f(v) \leq op_{t+1}(v)$.

Since in each case the contribution is not negative, the lemma follows. □

3.3. A New Upper Bound on the Maximum Transient Length

Now we present a tight upper bound on the transient length of f-reversible processes. Lemma 5 gives an upper bound based on the potential function and the cardinality of S_2 at $\tau(c_0) - 1$, the last time step of the transient phase.

Lemma 5. *For c_0 such that $\tau(c_0) > 0$, $\tau(c_0) \leq P(\tau(c_0)-1) - |S_2(\tau(c_0)-1)| + 1$. Moreover, this bound is attained.*

Proof. As in the proof of Lemma 4, if the process is not in the periodic phase at time t then there must exist at least one vertex v such that either $v \in (S_1(t) \cap S_2(t+1))$ or $v \in (S_2(t) \cap S_1(t+1))$. Let $T_{S_1 \to S_2}(v)$, $T^{>0}_{S_2 \to S_1}(v)$, and $T^{=0}_{S_2 \to S_1}(v)$ be the number of time steps in which v passes from $S_1(t)$ to $S_2\,(t+1)$, from $S_2(t)$ to $S_1(t+1)$ with potential variation, and from $S_2(t)$ to $S_1\,(t+1)$ without potential variation, respectively, for all $0 \leq t \leq \tau(c_0) - 2$. We denote $\sum_{v \in V(G)} T_{S_1 \to S_2}(v)$ by $T_{S_1 \to S_2}$ and, analogously, we define $T^{>0}_{S_2 \to S_1}$ and $T^{=0}_{S_2 \to S_1}$. Thus, for any initial configuration, the maximum transient length could be given in such a way just one vertex changes between S_1 and S_2, from t to $t+1$. Hence, it follows that

$$\tau(c_0) \;\leq\; T_{S_1 \to S_2} + T^{>0}_{S_2 \to S_1} + T^{=0}_{S_2 \to S_1} + 1. \tag{7}$$

Let $S_{i \to j}$ be the set of vertices v such that $v \in S_i(0)$ and $v \in S_j\,(\tau(c_0) - 1)$, for $i, j \in \{1, 2\}$. Now, we consider the relation between the numbers of transitions of a vertex v from time 0 to $\tau(c_0) - 1$:

$$T^{>0}_{S_2 \to S_1}(v) + T^{=0}_{S_2 \to S_1}(v) \;=\; \begin{cases} T_{S_1 \to S_2}(v) & \text{, if } v \in S_{1 \to 1}; \\ T_{S_1 \to S_2}(v) & \text{, if } v \in S_{2 \to 2}; \\ T_{S_1 \to S_2}(v) + 1 & \text{, if } v \in S_{2 \to 1}; \\ T_{S_1 \to S_2}(v) - 1 & \text{, if } v \in S_{1 \to 2}. \end{cases}$$

The above equation shows that

$$T^{>0}_{S_2 \to S_1} + T^{=0}_{S_2 \to S_1} \;=\; T_{S_1 \to S_2} + |S_{2 \to 1}| - |S_{1 \to 2}|. \tag{8}$$

By Eq. (7) and Eq. (8) we get that

$$\tau(c_0) \;\leq\; 2T_{S_1 \to S_2} + |S_{2 \to 1}| - |S_{1 \to 2}| + 1. \tag{9}$$

As in the proof of Lemma 4, for every $v \in (S_1(t) \cap S_2(t+1))$ a potential increase of at least 2 is obtained. Thus $T_{S_1 \to S_2} \leq (P\,(\tau(c_0) - 1) - P(0))\,/2$.

Furthermore, by Eq. (2), we get that $P(0) \geq |S_2(0)| = |S_{2\to1}| + |S_{2\to2}|$. Hence, we can rewrite Eq. (9) and the lemma follows:

$$\begin{aligned}\tau(c_0) &\leq P(\tau(c_0)-1) - P(0) + |S_{2\to1}| - |S_{1\to2}| + 1;\\ &\leq P(\tau(c_0)-1) - |S_{2\to1}| - |S_{2\to2}| + |S_{2\to1}| - |S_{1\to2}| + 1;\\ &= P(\tau(c_0)-1) - |S_2(\tau(c_0)-1)| + 1.\end{aligned}$$

□

Let us consider $V(G)$ partitioned according to time step $\tau(c_0)-1$ as follows:

- $X_1 = \{v \in V(G) : c_{\tau(c_0)-1}(v) = 0 \text{ and } v \in S_1(\tau(c_0)-1)\}$;
- $X_2 = \{v \in V(G) : c_{\tau(c_0)-1}(v) = 0 \text{ and } v \in S_2(\tau(c_0)-1)\}$;
- $Y_1 = \{v \in V(G) : c_{\tau(c_0)-1}(v) = 1 \text{ and } v \in S_1(\tau(c_0)-1)\}$;
- $Y_2 = \{v \in V(G) : c_{\tau(c_0)-1}(v) = 1 \text{ and } v \in S_2(\tau(c_0)-1)\}$.

Given disjoint subsets A and B of $V(G)$, we denote the set of all edges with one end in A and the other in B by $[A, B]$. Let us denote the minimum threshold value by $f_{\min}$. Furthermore, let us denote $S_f = \sum_{v\in V(G)} f(v)$. Thus, by Eq. (2) and Eq. (4), it follows that

$$\begin{aligned}P(\tau(c_0)-1) &= \sum_{v\in S_2(\tau(c_0)-1)} f(v) - \sum_{v\in S_1(\tau(c_0)-1)} f(v) + 2|[X_1,Y_1]| - 2|[X_2,Y_2]|\\ &= S_f - 2\left(\sum_{v\in S_1(\tau(c_0)-1)} f(v)\right) + 2|[X_1,Y_1]| - 2|[X_2,Y_2]|.\end{aligned} \tag{10}$$

Next, by Lemma 5 and Eq. (10) we obtain a tight upper bound on $\tau(c_0)$, for all c_0.

Theorem 6. *For c_0 such that $\tau(c_0) > 0$,*

$$\tau(c_0) \leq \begin{cases} S_f - (n + 2f_{\min} - 2) & , \text{if } X_1 = \emptyset \text{ or } Y_1 = \emptyset;\\ S_f - (n + 2f_{\min} - 1) & , \text{if } p(c_0) = 1,\ X_1 \neq \emptyset, \text{ and } Y_1 \neq \emptyset;\\ S_f + 2m - 3n + 1 - & \\ \quad \sum_{v\in S_1(\tau(c_0)-1)} (2f(v) - 3) & , \text{if } p(c_0) = 2,\ X_1 \neq \emptyset, \text{ and } Y_1 \neq \emptyset. \end{cases}$$

Proof. **Case 1:** $X_1 = \emptyset$ or $Y_1 = \emptyset$:

In this case we get that $[X_1, Y_1] = \emptyset$. Thus

$$P(\tau(c_0)-1) \leq S_f - 2\left(\sum_{v\in S_1(\tau(c_0)-1)} f(v)\right)$$

and, by Lemma 5, we have

$$\begin{aligned}
\tau(c_0) \leq\ & S_f - 2\left(\sum_{v\in S_1(\tau(c_0)-1)} f(v)\right) - (n - |S_1(\tau(c_0)-1)|) + 1 - 2|[X_2,Y_2]| \\
=\ & S_f - 2\left(\sum_{v\in S_1(\tau(c_0)-1)} f(v)\right) + |S_1(\tau(c_0)-1)| - n + 1 - 2|[X_2,Y_2]| \\
=\ & S_f - (n-1) - \sum_{v\in S_1(\tau(c_0)-1)} (2f(v)-1) - 2|[X_2,Y_2]|.
\end{aligned} \tag{11}$$

Since $S_f - (n-1)$ is a constant, $|S_1(\tau(c_0)-1)| \geq 1$, and $2f(v)-1 > 0$ for all $v \in V(G)$, the limit obtained by Eq. (11) is maximum when $[X_2,Y_2] = \emptyset$ and $S_1(\tau(c_0)-1) = \{v\}$, where that $f(v) = f_{\min}$. Hence, we obtain

$$\tau(c_0) \ \leq\ S_f - (n-1) - (2f_{\min}-1) \ =\ S_f - (n + 2f_{\min} - 2).$$

Case 2: $p(c_0) = 1$, $X_1 \neq \emptyset$, and $Y_1 \neq \emptyset$:
Since every $v \in S_1(\tau(c_0)-1)$ belongs to $S_2(\tau(c_0))$, we have that $|N_{Y_1}(v)| \leq f(v) - 1$, for every $v \in X_1$ and $|N_{X_1}(u)| \leq f(u) - 1$, for every $u \in Y_1$. Therefore

$$2|[X_1,Y_1]| \ =\ \sum_{v\in X_1} |N_{Y_1}(v)| + \sum_{u\in Y_1} |N_{X_1}(u)| \ \leq\ \sum_{v\in S_1(\tau(c_0)-1)} f(v) - |S_1(\tau(c_0)-1)|.$$

Thus, we can rewrite Eq. (10) as

$$\begin{aligned}
P(\tau(c_0)-1) \leq\ & \sum_{v\in S_2(\tau(c_0)-1)} f(v) - \sum_{v\in S_1(\tau(c_0)-1)} f(v) + \sum_{v\in S_1(\tau(c_0)-1)} f(v) - \\
& |S_1(\tau(c_0)-1)| - 2|[X_2,Y_2]| \\
\leq\ & \sum_{v\in S_2(\tau(c_0)-1)} f(v) - |S_1(\tau(c_0)-1)|.
\end{aligned}$$

By Lemma 5 and the fact that $|S_1(\tau(c_0)-1)| \geq 2$, it follows that

$$\begin{aligned}
\tau(c_0) \leq\ & \sum_{v\in S_2(\tau(c_0)-1)} f(v) - |S_1(\tau(c_0)-1)| - |S_2(\tau(c_0)-1)| + 1 \\
=\ & \left(\sum_{v\in V(G)} f(v) - \sum_{v\in S_1(\tau(c_0)-1)} f(v)\right) - n + 1 \\
\leq\ & S_f - (n + 2f_{\min} - 1).
\end{aligned}$$

Case 3: $p(c_0) = 2$, $X_1 \neq \emptyset$, and $Y_1 \neq \emptyset$:
As in Case 2, $P(\tau(c_0)-1)$ is maximum when $[X_2,Y_2] = \emptyset$, while $|[X_1,Y_1]| \leq m - |S_2(\tau(c_0)) - 1|$. It means that $S_1(\tau(c_0)-1)$ and $S_2(\tau(c_0)-1)$ induce

independent sets. Furthermore, $d(v) = 1$ for all $v \in S_2(\tau(c_0) - 1)$. Hence, we can rewrite Eq. (10) as

$$\begin{aligned}
P(\tau(c_0) - 1) \leq\ & S_f - 2\left(\sum_{v \in S_1(\tau(c_0)-1)} f(v)\right) + 2(m - |S_2(\tau(c_0) - 1)|) \\
&= S_f + 2m - \ 2\left(\sum_{v \in S_1(\tau(c_0)-1)} f(v)\right) - 2|S_2(\tau(c_0) - 1)|.
\end{aligned}$$

Hence, by Lemma 5 we complete the proof:

$$\begin{aligned}
\tau(c_0) \leq\ & S_f + 2m - 2\left(\sum_{v \in S_1(\tau(c_0)-1)} f(v)\right) - 3|S_2(\tau(c_0) - 1)| + 1 \\
=\ & S_f + 2m + 1 - 2\left(\sum_{v \in S_1(\tau(c_0)-1)} f(v)\right) - 3(n - |S_1(\tau(c_0) - 1)|) \\
\leq\ & S_f + 2m - 3n + 1 - \sum_{v \in S_1(\tau(c_0)-1)} (2f(v) - 3) \\
=\ & S_f + 2m - 3n + 1 - \sum_{v \in S_1(\tau(c_0)-1)} (2f(v) - 3).
\end{aligned}$$

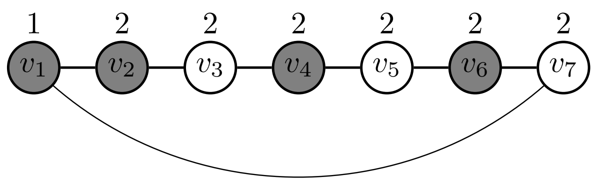
□

Fig. 3 shows an example illustrating that the bound on Case 1 of Theorem 6 is tight. The gray vertices have state 1 and the number above a vertex v_i represents $f(v_i)$. We consider an f-reversible process $R_f(C_n, c_0)$ on odd cycles $C_n = v_1v_2 \dots v_n$, in which $f(v_1) = 1$ and $f(v_i) = 2$, for all $i \neq 1$. Moreover $c_0(v_1) = 0$ if and only if $i > 1$ and odd. This process is uplifting and satisfies the following conditions:

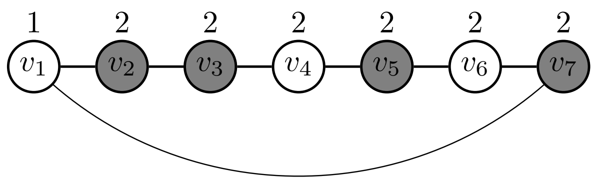
- $v_1 \in S_1(t)$, for all $t < \tau(c_0)$;
- For every $j > 1, v_j \in S_1(t)$, for all $t < (j - 2)$;
- For every $j > 1, v_j \in S_2(t)$, for all $t \geq (j - 2)$.

Thus $\tau(c_0) = n-1$, as well as in Case 1 of Theorem 6. However, such a bound can be arbitrarily far from $\tau(c_0)$. For instance, consider an f-reversible process on a star G with n vertices and whose central vertex is v. Suppose $f(v) = n$ and $f(u) = 1$, for all $u \neq v$. If $c_0(v) = 1 - c_0(u)$, for every $u \neq v$, then $\tau(c_0) = 1$, although Theorem 6 results on $\tau(c_0) \leq (n - 1)$. Notice that Lemma 5 gives to us the correct value.

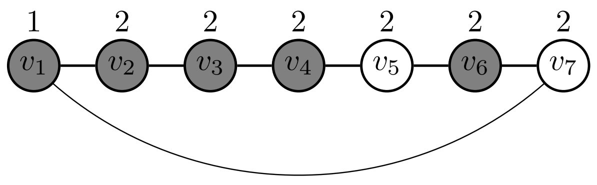
Figure 4e is an example attaining the bound on Case 2 of Theorem 6, where every vertex has threshold value 2. With respect to Case 3, we can cite an 1-reversible process on a path $P_n = \{v_1, v_2, \dots, v_n\}$ with $n \geq 3$, in which v_1 has the opposite state from the all others. Such a process has transient length equal to $n - 2$, as well as in Case 3.



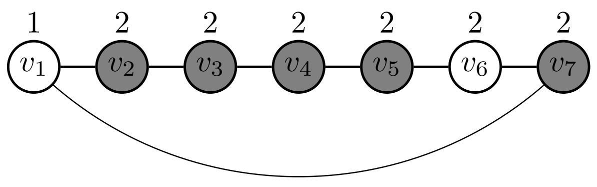
(a) c_0



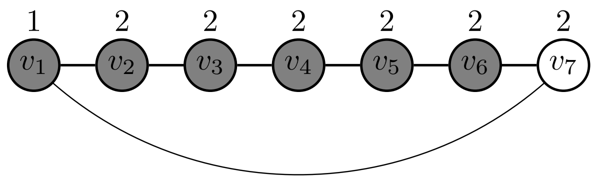
(b) c_1



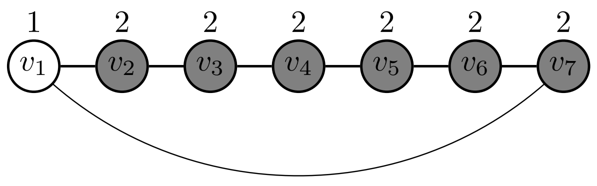
(c) c_2



(d) c_3



(e) c_4



(f) c_5

Figure 3: Transient phase of $R_f(C_7, c_0)$.

4. The Transient Length of 2-Reversible Processes on Trees

In this section we consider the maximum transient length of 2-reversible processes on trees. If $n \leq 4$ and $\tau(c_0) > 0$, it can be seen that $\tau(c_0) \leq n-2$, for all c_0. Such a limit holds when T is a path with 3 vertices, whose central vertex has the opposite state to the others (Fig. 4a). However, all non-periodic configurations with four vertices (Fig. 4) have transient length equal to $n-3$. Actually, we prove that $n-3$ is a tight upper bound on all initial configurations on trees with $n \geq 4$. If $n \leq 2$ then all configurations are periodic.

Theorem 7. *For T a tree with $n \geq 4$, $\tau_2(T) \leq n-3$.*

Proof. The theorem follows directly in both Cases 2 and 3 of Theorem 6, while Case 1 shows that $\tau_2(T) \leq n-2$. For trees T with $f(v) = 2$ for all $v \in V(T)$, we resort to Eq. (10) and Lemma 5 as follows, where $p = |S_1(\tau(c_0)-1)|$:

$$\begin{aligned}
\tau(c_0) &\leq S_f - 2\left(\sum_{v \in S_1(\tau(c_0)-1)} f(v)\right) + 2|[X_1,Y_1]| - 2|[X_2,Y_2]| - \\
&\qquad |S_2(\tau(c_0)-1)| + 1 \\
&\leq 2(n) - 2(2p) - (n-p) + 1 + 2|[X_1,Y_1]| - 2|[X_2,Y_2]| \\
&= n - 3p + 1 + 2|[X_1,Y_1]| - 2|[X_2,Y_2]|.
\end{aligned} \tag{12}$$

We split the proof into cases according to the cardinalities of sets X_1 and Y_1.
Case 1: $X_1 = \emptyset$ or $Y_1 = \emptyset$:

By Eq. (12) and the fact that $p \geq 1$, if $p > 1$ or $[X_2, Y_2] \neq \emptyset$ then $\tau(c_0) \leq n-4$. Therefore, let us suppose that $[X_2,Y_2] = \emptyset$ and $S_1(\tau(c_0)-1) = \{v\}$, where $N(v) = \{v_1, v_2, \ldots, v_{d(v)}\}$, $d(v) \geq 2$. Moreover, let us consider v being the root of T and $c_{\tau(c_0)-1}(v) = 0$, without loss of generality. Thus, each subtree T_{v_i}, rooted at v_i, has all its vertices with the same state at $\tau(c_0)-1$, for each $i \in \{1, 2, \ldots, d(v)\}$. Since only v must change its state at time $\tau(c_0)-1$, there must exist at least two subtrees T_{v_i} whose vertices have state 1 at time $\tau(c_0)-1$. Furthermore there exists at most one subtree with all its vertices having state 0 at time $\tau(c_0)-1$. Hence $p(c_0) = 1$ in this case.

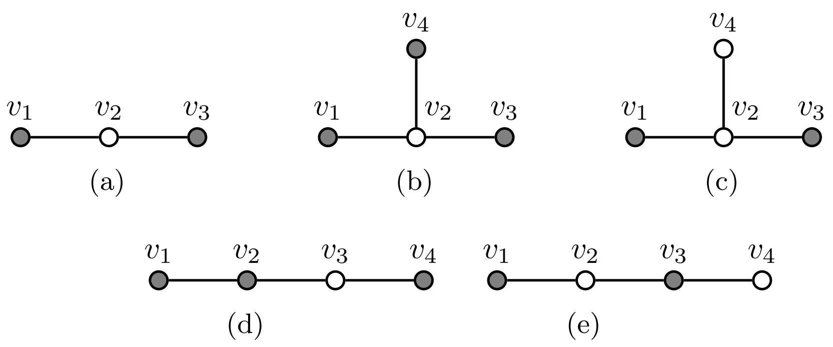


Figure 4: All non-periodic configurations with three and four vertices.

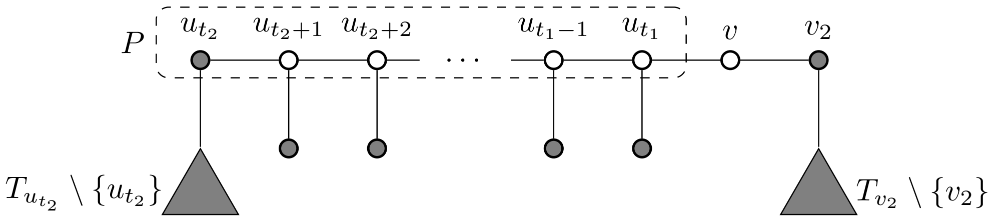


Figure 5: Representation of configuration c_{t_2} in Sub-case 1.1.

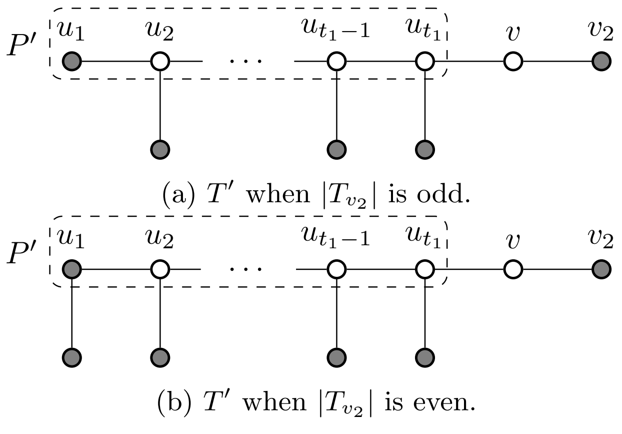
Let us consider that T_{v_1} is the last subtree whose vertices reach their final state. Since the process follows concurrently in all subtrees, we split the analysis in two sub-cases based on the degree of v.

Sub-case 1.1: $d(v) = 2$.

Notice that all vertices in $V(T) \setminus \{v\}$ must have the same state 1 at time $\tau(c_0) - 1$, where the process is uplifting from the leaves toward v.

Let t_1 and t_2 be the time steps in which T_{v_1} and T_{v_2} are uplifting, respectively. Since $t_1 = \tau(c_0) - 1$, we get that both of v and v_1 must keep state 0 from t_2 to t_1. If $t_1 = t_2$, then t_1 is maximum when $V(T_{v_1})$ induces a maximum path that is uplifting. Thus T_{v_1} is a path whose states of the vertices alternate. Furthermore T_{v_2} must have the same length as T_{v_1}. Thus T is an odd size path whose vertices alternate, implying that $\tau(c_0) = \frac{n-1}{2}$. Hence, if $\tau(c_0) = n - 2$ then $n = 3$, and if $\tau(c_0) = n - 3$ then $n = 5$. These cases are depicted in Fig. 4a and Fig. 7b, respectively.

On the other hand, if $t_1 > t_2$, it means that v_1 must keep state 0 from t_2 to $t_1 - 1$. Thus $t_1 - t_2$ is equal to the length of a path $P = \{u_{t_2}, u_{t_2+1}, \ldots, u_{t_1}\}$, where $u_{t_1} = v_1$ and each u_i reaches state 1 at time i, for all $i \in \{t_2, t_2+1, \ldots, t_1\}$. Moreover, for every u_i we get that $d(u_i) > 2$ and each one must have at most one neighbor with opposite state, from time t_2 to time i. The configuration obtained at time t_2 which maximizes $|P|$ is depicted in Fig. 5, where all vertices have state 1, unless v and those of $V(P) \setminus \{u_{t_2}\}$.



(a) T' when $|T_{v_2}|$ is odd.

(b) T' when $|T_{v_2}|$ is even.

Figure 6: Representation of tree T' obtained from a tree T, such that $\tau_2(T') \geq \tau_2(T)$.

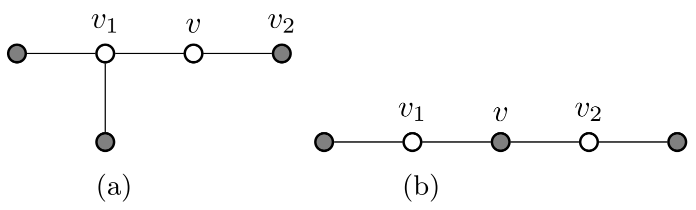


Figure 7: All configurations with $\tau(c_0) \geq n-3$, such that $X_1 = Y_1 = \emptyset$, $d(v) = 2$ and $n \geq 5$.

For every tree T, we can obtain a tree T' such that $\tau_2(T') \geq \tau_2(T)$ as follows. To each pair of vertices w and w' of $T_{v_2} \setminus \{v_2\}$, remove them from T_{v_2}, add w to P and, add an edge between w and w'. Moreover, assign $c_0(w) = 0$ and $c_0(v_2) = c_0(w') = 1$. If $|T_{v_2}|$ is even then there remain two vertices in T_{v_2}, by the previous procedure. Thus, remove the last neighbor w'' of v_2 from T_{v_2} and add it to T_{v_1} as a neighbor of the vertex of P at maximum distance from v, such that $c_0(w'') = 1$. Fig. 6a and Fig. 6b represent the cases in which $|T_{v_2}|$ is odd or even, respectively. Thus path P increases, yielding a new path $P' = \{u_1, u_2, \ldots, u_{t_1}\}$. Hence it is obtained one more time for each pair of moved vertices.

Note that $\tau_2(c_0) = |P'|$. Hence, if $|T_{v_2}|$ is odd then $\tau(c_0) = |P'| \leq \frac{n-1}{2}$ and it follows that:

- $\frac{n-1}{2} = n-2 \;\Rightarrow\; n = 3$ and $|P'| = 1$ (Fig. 4a);
- $\frac{n-1}{2} = n-3 \;\Rightarrow\; n = 5$ and $|P'| = 2$ (Fig. 7a).

Finally, if $|T_{v_2}|$ is even then $\tau(c_0) = |P'| = \frac{n-2}{2}$ and it follows that:

- $\frac{n-2}{2} = n-2 \;\Rightarrow\; n = 2$ and $|P'| = 0$;
- $\frac{n-2}{2} = n-3 \;\Rightarrow\; n = 4$ and $|P'| = 1$ (Fig. 4d).

Sub-case 1.2: $d(v) \geq 3$.

Let us suppose that all non-leaf vertices reach their final states exactly one time step after all of their children. Let $P = \{p_1, p_2, \ldots, p_\ell\}$ be a longest path from $p_1 = v$ until a leaf p_ℓ. Thus, for each internal vertex p_i, we get that p_i reaches its final state exactly after p_{i+1}, $i < \ell$ and $\tau(c_0) = |P|$, where $|P| \leq n-3$. Fig. 8 depicts this case, in which $d(v) = 3$ and the internal vertices of P must alternate their states, since they have degree equal to 2. Therefore v must have 2 neighbors with opposite states, for all $t < \tau(c_0) - 1$. If $d(v) \geq 4$ or $d(p_i) \geq 3$ for some internal vertex p_i then $\tau(c_0) \leq n-4$, in this case.

Now, let $u \neq v$ be a non-leaf vertex whose state does not change even if all of its children reach their final states. Moreover, let consider u the farther vertex from v satisfying this property. Since all leaves of a subtree T_{v_i} must have the same state, say $s \in \{0, 1\}$, we have $d(u) = 2$. Moreover u and its parent w must have the same initial state 0, when the child of u achieves its final state, and u changes its state exactly one time step after w. In other words, the uplifting of T_{v_i} follows from the leaves to v, but it stops at u, which is "released" by w.

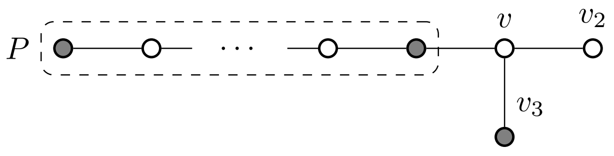
Therefore, either w is released by at least two children, or w also depends on its parent. Hence, there exists a path $W = \{w_x, w_{x-1}, \dots, w_1\}$, such that w_i depends on its parent w_{i-1} to change its state, for all $i \in \{1, \dots, x-1\}$. Thus we get that $d(w_i) \geq 3$, for all $w_i \in W$. Moreover w_1 is the first vertex of W to be released, where the vertices in $W \cup \{u\}$ change their initial states from w_1 to u. If $z_1 = N(w_1) \cap P$ does not reach its final state before w_1, the process follows changing the states of the vertices from w_1 to u and returns to w_1. Otherwise, the process takes fewer time steps, since it would follow simultaneously to $T \setminus T_{z_1}$. In this case, all vertices on path $Z = \{z_1, z_2, \dots, z_y\}$ must alternate their states until T_{w_1} is uplifting.

Let $T'(w_i)$ be the subtree of w_i which does not intersect P, for all $i \in \{1, \dots, x\}$. To maximize the transient length, w_i must keep state 0 from $t = 0$ until its parent and all of its children achieve state 1. Thus w_i waits until T'_{w_i} is uplifting (see Fig. 9). Moreover, since w_i does not affect the state of its children, the maximum number of time steps required to the uplifting of T'_{w_i} is $|T'_{w_i}| - 1$. Thus, it follows that

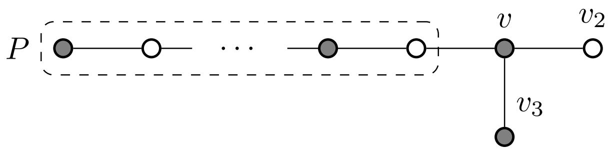
$$\begin{aligned}
\tau(c_0) &\leq 2x + y + 2 + \sum_{i=1}^{x} \tau_2(T'_{w_i}) + \tau_2(T_u) \\
&\leq 2x + y + 2 + \sum_{i=1}^{x} (|T'_{w_i}| - 1) + (|T_u| - 1) \\
&= x + y + 1 + \sum_{i=1}^{x} |T'_{w_i}| + |T_u| \\
&\leq n - 3.
\end{aligned}$$

Thus $\tau(c_0) = n - 3$ when T is as in Fig. 10a and Fig. 10b, where all subtrees T'_{w_i} and T_u have exactly one vertex and all vertices of Y have degree 2.

Note that if $x = 0$ then the obtained configurations are equivalent to those in Fig. 8. On the other hand, if n is even and $y = 0$ then v "starts" the process,



(a) n odd.



(b) n even.

Figure 8: Representation of the initial configurations of trees in which $\tau_2(c_0) = n - 3$ and all non-leaf vertices reach their final states exactly one time step after all of their children.

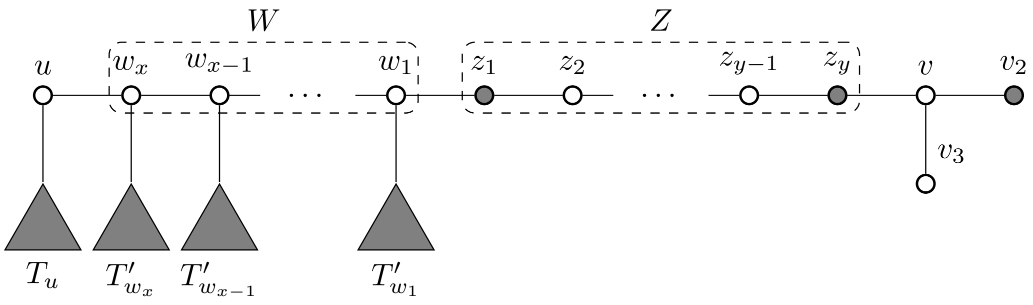
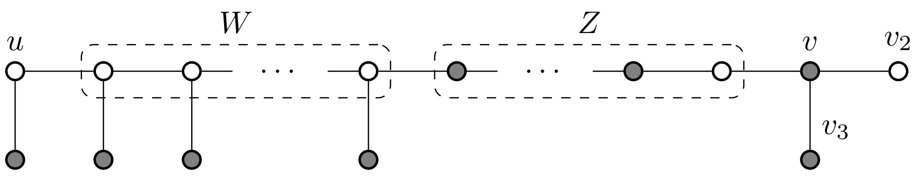
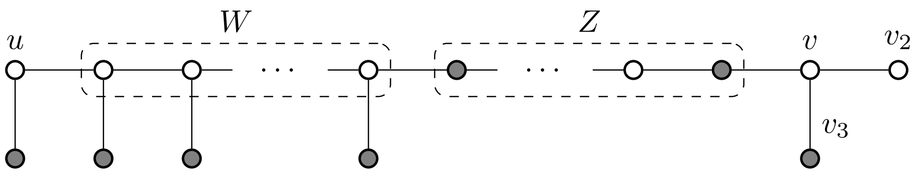


Figure 9: Representation of tree T, where $v \in S_1(t)$, for all $t \leq \tau(c_0) - 1$.

where its state must be equal to the states of the leaves of P, as well as when n is odd and $y = 0$, but with initial state opposite to that of the leaves. In the last case it is possible to keep the state of v, adding one neighbor to v with opposite state, yielding a tree with even number of vertices (Fig. 11a). The effect is to extend W including v, increasing the transient length by one time step and the number of vertices also by one, keeping the upper bound on $n-3$. On the other hand, no other vertex can be added as a neighbor of v, since v already have two neighbors of each state. Finally, we can obtain the configuration given in Fig. 11b, where the additional vertex is added as a neighbor of v_2, in which the effect is the same as the previous case. Moreover, any additional vertex does not increase the transient length. Hence, no other initial configuration is possible.



(a) n even.



(b) n odd.

Figure 10: Representation of a general initial configuration of trees in which $\tau(c_0) = n - 3$ and at least one non-leaf vertex reaches its final state after all of its children.

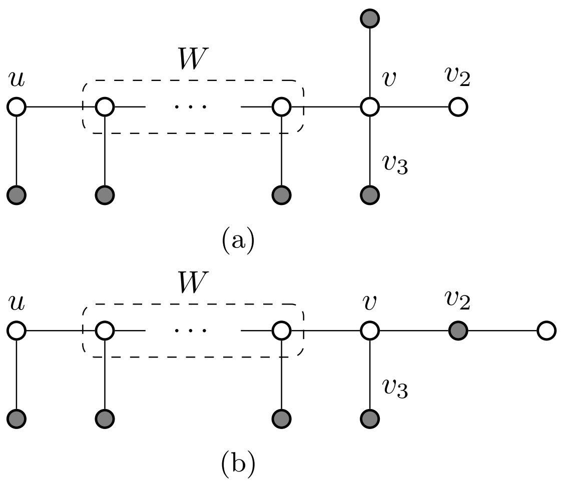


Figure 11: Representation of trees T with $\tau(c_0) = n - 3$, in which $y = 0$ and n even.

Case 2: $X_1 \neq \emptyset$ and $Y_1 \neq \emptyset$:

Since $|[X_1, Y_1]| \leq p - 1$ and $V(S_1(\tau(c_0) - 1))$ induces a forest, by Eq. (12) it follows that

$$\begin{aligned}\tau(c_0) \leq\ & n - 3p + 1 + 2(p-1) - 2|[X_2, Y_2]| \\ =\ & n - p - 1 - 2|[X_2, Y_2]|.\end{aligned}$$

Therefore, if $p > 2$ or $[X_2, Y_2] \neq \emptyset$ then $\tau(c_0) \leq n - 4$ and the theorem follows. Thus, let us suppose $[X_2, Y_2] = \emptyset$ and consider $S_1(\tau(c_0) - 1) = \{u, v\}$, such that $(u, v) \in E(T)$ and $c_{\tau(c_0)-1}(v) = 1 - c_{\tau(c_0)-1}(u)$.

Since every $w \in S_2(\tau(c_0) - 1)$ has at most one neighbor with opposite state (u or v), w belongs to $S_2(\tau(c_0))$. Thus $|S_1(\tau(c_0))| = 2$ only if $S_1(\tau(c_0)) = \{u, v\}$, implying that $c_{\tau(c_0)-1}$ is periodic, a contradiction. Therefore, either $p(c_0) = 1$, where u and v have opposite states, or $S_1(\tau(c_0))$ contains exactly one vertex, which is either u or v.

Suppose $p(c_0) = 2$, where $S_1(\tau(c_0)) = v$ and $c_{\tau(c_0)-1}(v) = i$, $i \in \{0, 1\}$. Thus $d(v) \geq 4$ and v has at least one neighbor with state i and at least three with state $1 - i$ (where u is one of them), at $\tau(c_0) - 1$. Since $(N(u) \setminus \{v\}) \subset S_2(\tau(c_0) - 1)$, it follows that the value of $\tau(c_0)$ is at most the maximum length of a path P whose vertices alternate their states, where P is a subtree of u. Hence $\tau(c_0) \leq n - |N(v) \cup \{v\}| = n - 5$.

Now, suppose that $p(c_0) = 1$. Thus $c_{\tau(c_0)-1}(w) = 1 - c_{\tau(c_0)-1}(u)$, for all $w \in N(u)$, and $c_{\tau(c_0)-1}(w') = 1 - c_{\tau(c_0)-1}(v)$, for all $w' \in N(v)$. Therefore $\tau(c_0)$ is maximum when a subtree T'_u of u has maximum transient length. Thus T'_u is a path whose vertices alternate their states. Hence, we get that $\tau(c_0) \leq n - |N(v) \cup \{v\}| \leq n - 3$. Moreover, the previous limit is obtained only if $d(u) = d(v) = 2$, where $\tau(c_0) = 1$, since $v \in S_2(1)$. This situation is depicted in Fig. 4e. Finally, consider $d(u) \geq 3$. Since $V(G) \setminus \{u, v\} = S_2(\tau(c_0) - 1)$, all subtrees of u which do not contain v and every subtree of v that do not contain u must reach their final states at same time step $\tau(c_0) - 1$. Analogously, $\tau(c_0)$ is maximum when a subtree T'_v of v is a path whose vertices alternate their states. Hence $\tau(c_0) \leq n - |N(u) \cup \{u, v\}| = n - 4$. ☐

Let $\#\tau_f(G, q)$ be the number of configurations c_0 such that $\tau(c_0) = q$ for an f-reversible process on G. Thus, we prove that $\#\tau_2(T, n-3) = O(n)$ for trees T with $n \geq 4$.

Corollary 8. *For T a tree with $n \geq 4$,*

$$\#\tau_2(T, n-3) = \begin{cases} 4 & , \textit{if } \ n = 4; \\ 3 & , \textit{if } \ n = 5; \\ \frac{n}{2} & , \textit{if } n \geq 6 \textit{ and } n \textit{ is even}; \\ \frac{n-3}{2} & , \textit{if } \ n \geq 7 \textit{ and } n \textit{ is odd}. \end{cases}$$

Proof. Fig. 4 presents all initial configurations when $n = 4$. As in the proof of Theorem 7, there are only three initial configurations attaining the bound $n-3$ when $n = 5$, where two of them are depicted in Fig. 7 and the last one is given by Fig. 8a. For $n = 6$, since $d(v) \geq 3$ (where $S_1(\tau(c_0) - 1) = \{v\}$) each subtree of v has at most three vertices. Thus, there is no configuration with $|W| > 0$. Hence, either $|Z| = 0$ or $|Z| = 3$. When $|Z| = 0$ we get that both configurations attaining the bound $n-3$ are given in Fig. 11a and Fig.11b. Fig. 8b represents the case in which $|Z| = 3$.

If $n \geq 7$ and n is odd then $|Z|$ must be even and there exists exactly one configuration for each even value of $|Z|$ from 0 to $n-7$. Furthermore, there exists one more for $|Z| = n-3$. Hence, we get that

$$\#\tau_2(T, n-3) = 1 + \sum_{i=0 \text{ and } i \text{ even}}^{n-7} (1) = 1 + \left(\frac{n-5}{2}\right) = \frac{n-3}{2}.$$

Analogously, if $n \geq 8$ and n is even then for the cases in which $|Z| > 0$, we get that $|Z|$ must be odd and there exists exactly one configuration for each odd value of $|Z|$ from 1 to $n-7$. Moreover, there are two configurations when $|Z| = 0$ (Fig. 11) and one more when $|Z| = n-3$ (Fig. 8a). Hence, we get that

$$\#\tau_2(T, n-3) = 3 + \sum_{i=1 \text{ and } i \text{ odd}}^{n-7} (1) = 3 + \left\lceil \frac{n-7}{2} \right\rceil = 3 + \left(\frac{n-6}{2}\right) = \frac{n}{2}.$$

☐

Corollary 9. *For T a tree with $n \geq 4$, $\tau_2(T) = n-3$ if and only if c_0 is output by Algorithm 1.*

Proof. In Algorithm 1, such trees T_j with $n \geq 4$ and initial configurations C_j, where $\tau_2(C_j) = n-3$, are as in the proof of Theorem 7. In such an algorithm, lines 1–12 represent the construction of the configuration represented in Fig. 8a and Fig. 8b for n odd and even, respectively. Lines 4–7 result on the construction of the tree, while lines 8–12 represent the assignment of the states to the vertices.

Thus, tree T_1 and configuration C_1 are generated. For trees with $n = 4$ or $n = 5$, lines 17–19 construct the trees and configurations given by Fig. 4b and Fig. 7a for $n = 4$ and $n = 5$, respectively. Lines 23–27 describe the construction of trees and configurations depicted by Fig. 4d and Fig. 7b for $n = 4$ and $n = 5$, respectively. Lines 31–34 we get the tree and configuration given by Fig. 4e. For $n \geq 6$, lines 36–48 modify the tree and configuration generated one step before, which are respectively T_1 and C_1, for each odd i from 3 to $n - 3$. At each step, the graph obtained has $|Z|$ decreased by 1, while $|W|$ increase by 1. Such configurations are represented by Fig. 10a and Fig. 10b for n even and odd, respectively. If n is even and $i = n-3$ then we obtain the configuration depicted in Fig. 11a. Lines 43–48 construct the tree and configuration depicted in Fig. 11b when n is even. Since all configurations c_0 attaining $\tau(c_0) = n - 3$ presented in Theorem 7 are generated by the algorithm and no other configuration can be obtained, the corollary follows. □

5. NP-Completeness of f-Conversion Set

In this section, we prove the NP-completeness for bipartite graphs of f-Conversion Set. The proof is a reduction through a restriction of 3SAT, where each clause has 2 or 3 literals and each variable occurs in at most 3 clauses [18]. It can be polynomially solved if all clauses have exactly 2 or exactly 3 literals [38]. Given an integer $q > 0$, Dourado *et al.* [12] proved that determining if $r_2(G) \leq q$ is NP-hard. Denoting the neighborhood of v in a vertex set X by $N_X(v)$, we say that two disjoint vertex subsets A and B are a *bad pair* of an f-reversible process $R_f(G, c_0)$ if:

- each vertex of A has a different state from the vertices of B;
- $|N_B(v)| \geq f(v)$, for all $v \in A$;
- $|N_A(u)| \geq f(u)$, for all $u \in B$.

As proved by Dourado *et al.* [12], every uplifting f-reversible process cannot contain any bad pair.

Lemma 10. *If $R_f(G, c_0)$ is uplifting and, for some $v \in V(G)$, $f(v) = d(v)$ and $f(u) = 1$ for every $u \in N(v)$, then $c_t(v) = 1$ and $c_t(u) = 1$ for some $u \in N(v)$ and every $t \geq 0$.*

Proof. Suppose by contradiction that the lemma is false. Since $R_f(G, c_0)$ is uplifting, if there exists a time step $t_0 \geq 0$ such that $c_{t_0}(v) = 0$ then there exists a time step $t > t_0$ in which v changes its state. Suppose t to be the smallest. In this case, $c_{t-1}(v) = 0$ and $c_{t-1}(u) = 1$ for all $u \in N(v)$. Hence, sets $A = \{v\}$ and $B = N(v)$ are a bad pair of $R_f(G, c_0)$, a contradiction. The argument for the existence of a vertex $u \in N(v)$ which has state 1 through the process is similar. □

Theorem 11. *f-Conversion Set is NP-complete for G bipartite with maximum degree 3 and $\mathrm{Im}_f = \{1, 2, 3\}$.*

Algorithm 1: Generate all trees T with $n \geq 4$ and corresponding initial configurations leading to $\tau_2(T) = n - 3$.

```
Input: Number $n$ of vertices.
Output: Trees $T_j$, each with the corresponding initial configuration $C_j$ such that
        $\tau(C_j) = n - 3$.
1  $V \leftarrow \{v_1, v_2, \dots, v_n\}$;
2  $E \leftarrow \emptyset$;
3  $j \leftarrow 1$;
4  for $i \leftarrow 1$ to $n - 2$ do                    // Lines 4 -- 7 result on tree $T_1$.
5      $E \leftarrow E \cup (v_i, v_{i+1})$;
6      if $i = n - 2$ then
7          $E \leftarrow E \cup (v_i, v_{i+2})$;
8  for $i \leftarrow 1$ to $n$ do                        // Lines 8 -- 12 result on configuration $C_1$.
9      if $i$ is odd then
10         $c_0(v_i) \leftarrow 1$;
11     else
12         $c_0(v_i) \leftarrow 0$;
13 $T_j \leftarrow G(V, E)$;
14 $C_j \leftarrow c_0$;          // Fig. 8a and Fig. 8b are obtained for $n$ odd and even, respectively.
15 $j \leftarrow j + 1$;
16 if $n = 4$ or $n = 5$ then
17     $c_0(v_4) \leftarrow 1 - c_0(v_4)$;       // Lines 17 -- 21 result on Fig. 4b and Fig. 7a for $n = 4$ and $n = 5$,
       respectively.
18     if $n = 5$ then
19         $c_0(v_3) \leftarrow 1 - c_0(v_3)$;
20     $T_j \leftarrow G(V, E)$;
21     $C_j \leftarrow c_0$;
22     $j \leftarrow j + 1$;
23     $E \leftarrow E \setminus (v_n, v_{n-2})$; // Lines 23 -- 29 result on Fig. 4d and Fig. 7b for $n = 4$ and $n = 5$, respectively.
24     $E \leftarrow E \cup (v_{n-1}, v_n)$;
25     if $n = 5$ then
26         $c_0(v_3) \leftarrow c_0(v_4)$;
27         $c_0(v_4) \leftarrow 1 - c_0(v_4)$;
28     $T_j \leftarrow G(V, E)$;
29     $C_j \leftarrow c_0$;
30     if $n = 4$ then                  // Lines 31 -- 34 obtain the tree and configuration given by Fig. 4e.
31         $j \leftarrow j + 1$;
32         $c_0(v_4) \leftarrow 1 - c_0(v_4)$;
33         $T_j \leftarrow G(V, E)$;
34         $C_j \leftarrow c_0$;
35 else
36     for $i \leftarrow 3$ to $n - 3$ do  // Lines 36 -- 42 obtain Fig. 10a and Fig. 10b for $n$ even and odd, respectively.
37         if $i$ is odd then                // For $n$ even and $i = n - 3$, it is obtained Fig. 11a.
38             $E \leftarrow E \setminus (v_i, v_{i-1})$;
39             $E \leftarrow E \cup (v_{i-1}, v_{i+1})$;
40             $T_j \leftarrow G(V, E)$;
41             $C_j \leftarrow c_0$;
42             $j \leftarrow j + 1$;
43         if $i = n - 3$ and $n$ is even then            // Lines 43 -- 48 obtain Fig. 11b for $n$ even.
44             $E \leftarrow E \setminus (v_{i+1}, v_{i+3})$;
45             $E \leftarrow E \cup (v_i, v_{i+3})$;
46             $T_j \leftarrow G(V, E)$;
47             $C_j \leftarrow c_0$;
48             $j \leftarrow j + 1$;
```

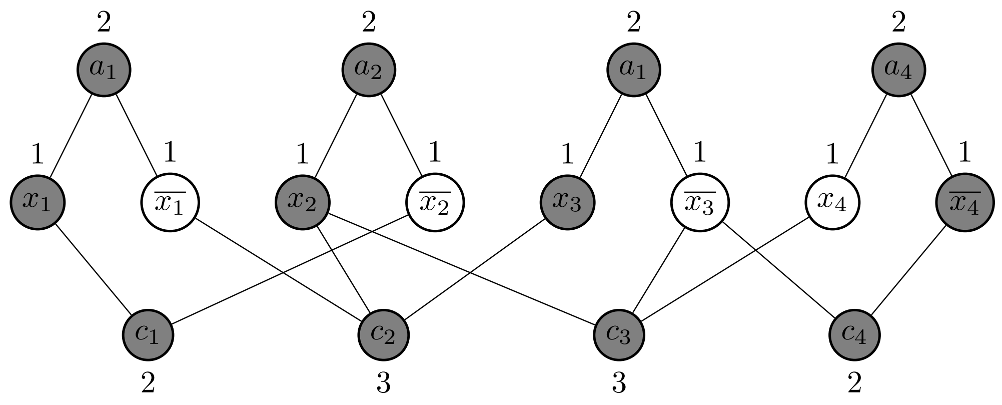


Figure 12: A labelling with $r_f(G)$ vertices satisfying $F = (X_1 \vee \overline{X_2}) \wedge (\overline{X_1} \vee X_2 \vee X_3) \wedge (X_2 \vee \overline{X_3} \vee X_4) \wedge (\overline{X_3} \vee \overline{X_4})$.

Proof. f-Conversion Set is in NP because, given an integer $q > 0$, we can execute the process from c_0, that has q vertices with state 1, to $c_{\tau(c_0)}$. Moreover, we can obtain c_{t+1} from c_t in $O(n+m)$ for all $t \geq 0$, because by Case 1 of Theorem 6 we can execute the whole process in $O((n+m)(S_f - n))$.

Let F be an instance of the restricted 3SAT problem with n variables $X_1, X_2, \ldots, X_n$ and m clauses $C_1, C_2, \ldots, C_m$. We construct a graph G of f-Conversion Set as follows. For each variable X_i, G contains three vertices denoted by x_i, $\overline{x_i}$ and a_i, $1 \leq i \leq n$. For each clause C_j, there is one vertex denoted by c_j, $1 \leq j \leq m$. Each vertex a_i is adjacent only to x_i and $\overline{x_i}$, $1 \leq i \leq n$. Moreover, edges $x^i c_j$, $x^i \in \{x_i, \overline{x_i}\}$, are added if and only if $x^i \in C_j$ in F, for all $1 \leq i \leq n$ and $1 \leq j \leq m$. Fig. 12 depicts the graph, initial configuration and threshold values obtained from $F = (X_1 \vee \overline{X_2}) \wedge (\overline{X_1} \vee X_2 \vee X_3) \wedge (X_2 \vee \overline{X_3} \vee X_4) \wedge (\overline{X_3} \vee \overline{X_4})$.

Let $A = \bigcup a_i$, $X = \bigcup \{x_i \cup \overline{x_i}\}$ and, $C = \bigcup c_i$ denote the sets of vertices called *auxiliar*, *literal* and, *clause* vertices, respectively. Note that each vertex of A has degree 2 and, since each clause has 2 or 3 literals and each literal occurs in 1 or 2 clauses, the maximum degree of G is 3. Moreover, $A \cup C$ and X are a bipartition of $V(G)$, where each part induces an independent set. Therefore G is a bipartite graph with maximum degree 3. To complete the transformation, we assign $f(x) = 1$, for each $x \in X$, and $f(v) = d(v)$ for all $v \in V(G) \setminus \{X\}$.

Now, we prove that F is satisfying if and only if $r_f(G) = m + 2n$. By Lemma 10, we get that all vertices $v \in A \cup C$ and at least one of x_i or $\overline{x_i}$ must have state 1 in c_0, for all $1 \leq i \leq n$. Thus $r_f(G) \geq m + 2n$. We can consider that $c_0(x^i) = 1$ if and only if x^i has its true value in F, $x^i \in \{x_i, \overline{x_i}\}$.

If F is satisfying then for each vertex c_j, $1 \leq j \leq m$, there is at least a neighbor that is a literal vertex with initial state 1. Furthermore, for each pair of literal vertices x_i and $\overline{x_i}$, $1 \leq i \leq n$, only one of them has its initial state equal to 1. Thus, clearly $c_1(v) = 1$ for every $v \in V(G)$ and $r_f(G) = m + 2n$. Now, if $r_f(G) = m + 2n$ then, for each pair of vertices x_i and $\overline{x_i}$, only one of them must have its initial state equal to 1, because all vertices $u \in A \cup C$

have their initial state equal to 1. Moreover, since the process is uplifting, each clause vertex c_j, $1 \leq j \leq m$, is adjacent to a literal vertex with initial state 1, by Lemma 10. Hence, all clauses are satisfied, concluding the proof. $\square$

We also analyze the complexity of finding a minimum f-conversion set on general graphs when $f(v) = d(v)$, for all $v \in V(G)$. In this case, if an edge is such that the states of its ends are the same then these vertices will never change their states, showing that $r_f(G) \geq \beta(G)$. We establish the following result relating $r_f(G)$ and $\beta(G)$ in this situation.

Theorem 12. *Let $R_f(G, c_0)$ be such that $f(v) = d(v)$ for all $v \in V(G)$. If a minimum vertex cover of G exists that is not an independent set then $r_f(G) = \beta(G)$. Otherwise, $r_f(G) = \beta(G) + 1$.*

Proof. Let C be a minimum vertex cover of G and let us consider that a vertex v belongs to C if and only if $c_0(v) = 1$. Clearly $r_f(G) \geq \beta(G)$, otherwise, there must exist at least one edge whose ends have state zero in c_0, meaning that the process does not uplift. Let I_t be the set of vertices v such that $c_t(v) = 0$, for all $0 \leq t \leq \tau(c_0) - 1$. Hence, I_t must induce an independent set in G and $r_f(G)$ is obtained taking I_0 the greatest possible.

Suppose that C does not induce an independent set in G. Thus there is at least one edge $e = (u, v)$ in $G[C]$ and, hence, u and v remain with state 1 forever. Suppose by contradiction that the process is not uplifting. Let us also suppose that there exists an edge e' whose ends have state zero in $c_{\tau(c_0)}$. Furthermore, let $t > 0$ be the first time step in which such an edge e' appears. Since at time $t - 1$ there were not any such edges, all vertices $z \in I_{t-1}$ change their states. Thus, I_t induces an independent set, a contradiction. Hence, if $p(c_0) = 1$, then the process is uplifting.

Thus, suppose that $p(c_0) = 2$. In this case, there must exist a bad pair A and B. Hence A and B induce independent sets, and since no edge whose ends have state zero is formed through the process, we get that $\tau(c_0) = 0$ and $G[A \cup B]$ is a connected component which does not contain u and v, a contradiction.

Finally, suppose that any minimum vertex cover C of G induces an independent set. Thus, we need more vertices than $\beta(G)$ for the uplifting. It is enough to show that we need only one vertex more than a minimum covering. Let $C \cup \{v'\}$ be the vertex subset of G with initial state 1, for some $v' \in V(G) \setminus C$. Hence, since G is connected, there is an edge in $G[C \cup \{v'\}]$ and the same argument of the previous case works now, completing the proof. $\square$

Corollary 13. *If G is not bipartite and $f(v) = d(v)$ for all $v \in G$ then $r_f(G) = \beta(G)$.*

Proof. Since there is no minimum vertex cover of G inducing an independent set, the corollary follows from Theorem 12. $\square$

6. Conclusions

We have presented a potential function to aid in the study of the periodic behavior of f-reversible processes, which we have proved to be of threshold networks. We have shown that this function is nondecreasing and, using this fact, we have presented new upper bound on the transient length based on f, n and m. Moreover, the potential function gave the necessary interpretation in other to obtain all initial configurations on 2-reversible processes on trees reaching at least $n-3$ time steps, where its value is tight when $n \geq 4$. Determining such configurations based on only G and f is a no trivial work. This can be observed by the work of Oliveira, Barbosa and Protti [36], in which determining if there exists a predecessor configuration of a given one is NP-complete even for bipartite graphs. In this way, we left two problems regarding the maximum transient length:

- To characterize all initial configurations reaching the maximum transient length in terms of only G and f;
- To count the number of such configurations.

We also have studied the problem of determining the minimum size $r_f(G)$ of a vertex subset that allows f-reversible processes to uplift. We have shown that determining if $r_f(G)$ is at most a constant $q > 0$ is NP-complete for bipartite graphs with maximum degree 3 and $\text{Im}_f = \{1, 2, 3\}$. We also have proved that $\beta(G) \leq r_f(G) \leq \beta(G) + 1$, when $f(v) = d(v)$ for all vertices. In fact, determining $r_f(G)$ seems a no trivial job even for simple graph classes as paths and cycles [12]. In addition to this open problem, we propose the following:

- To determine a lower and an upper bound on $r_f(G)$;
- To find an approximative algorithm to compute $r_f(G)$.

Acknowledgments

The authors acknowledge partial support from CNPq, CAPES, and FAPERJ BBP grants.

References

[1] J. Adler. Bootstrap percolation. *Physica A*, 171:453–470, 1991.

[2] Z. Agur. Fixed points of majority rule cellular automata with application to plasticity and precision of the immune system. *Complex Syst.*, 5:351–357, 1991.

[3] J.-P. Allouche, M. Courbage, and G. Skordev. Notes on cellular automata. *Complex Syst.*, 3:213–244, 2001.

[4] P. Balister, B. Bollobás, J. R. Johnson, and M. Walters. Random majority percolation. *Random Struct. Algorithms*, 36(3):315–340, 2010.

[5] C. L. Barrett, H. B. Hunt III, M. V. Marathe, S. S. Ravi, D. J. Rosenkrantz, and R. E. Stearns. Complexity of reachability problems for finite discrete dynamical systems. *J. of Computer Syst. Sci.*, 72(8):1317 – 1345, 2006.

[6] A. Bondy and U. Murty. *Graph Theory*. Graduate Texts in Mathematics. Spring-Verlag Press, 2008.

[7] Y. Caro and R. Pepper. Dynamic approach to k-forcing. *Theory and Applications of Graphs*, 2(2):1 – 5, 2015. Available at: http://digitalcommons.georgiasouthern.edu/tag/vol2/iss2/2.

[8] C. C. Centeno, M. C. Dourado, L. D. Penso, D. Rautenbach, and J. L. Szwarcfiter. Irreversible conversion of graphs. *Theor. Comput. Sci.*, 412(29):3693 – 3700, 2011.

[9] N. Chen. On the approximability of influence in social networks. *SIAM J. Discret. Math.*, 23:1400–1415, 2009.

[10] M. H. DeGroot. Reaching a consensus. *J. Am. Stat. Assoc.*, 69:167–182, 1974.

[11] P. Domingos and M. Richardson. Random majority percolation. *Proc. 7th ACM SIGKDD Int. Conf. Knowl. Discov. Data Min.*, pages 57–67, 2001.

[12] M. Dourado, L. Penso, D. Rautenbach, and J. Szwarcfiter. Reversible iterative graph processes. *Theor. Comput. Sci.*, 460:16–25, 2012.

[13] P. A. . Dreyer Jr. and F. S. Roberts. Irreversible k-threshold processes: Graph-theoretical threshold models of the spread of disease and of opinion. *Discret. Appl. Math.*, 157(7):1615 – 1627, 2009.

[14] P. A. Dreyer Junior. *Application and Variations of Domination in Graphs*. Ph.D. dissertation, The State University of New Jersey, New Brunswick, NJ, 2000.

[15] P. Flocchini, F. Geurts, and N. Santoro. Optimal irreversible dynamos in chordal rings. *Discret. Appl. Math.*, 113(1):23 – 42, 2001. Selected Papers: 12th Workshop on Graph-Theoretic Concepts in Computer Science.

[16] P. Flocchini, R. Královič, P. Ružička, A. Roncato, and N. Santoro. On time versus size for monotone dynamic monopolies in regular topologies. *J. Discret. Algorithms*, 1(2):129 – 150, 2003. {SIROCCO} 2000.

[17] French Jr. and J. R. P. A formal theory of social power. *Psychol. Rev.*, 63:181–194, 1956.

[18] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., San Francisco, CA, 1979.

[19] L. Gargano, P. Hell, J. G. Peters, and U. Vaccaro. Influence diffusion in social networks under time window constraints. *Theor. Comput. Sci.*, 584:53 – 66, 2015. Special Issue on Structural Information and Communication Complexity.

[20] D. C. Ghiglia, G. A. Mastin, and L. A. Romero. Cellular automata method for phase unwrapping. *J. Opt. Soc. Am. A*, 4:267–280, 1987.

[21] E. Goles and S. Martínez. *Neural and automata networks: dynamical behavior and applications*, volume 58 of *Mathematics and Its Applications*. Springer Netherlands, 2013.

[22] E. Goles and J. Olivos. Comportement périodique des fonctions á seuil binaires et applications. *Discrete Appl. Math.*, 3(2):93 – 105, 1981.

[23] E. Goles and J. Olivos. The convergence of symmetric threshold automata. *Inform. Control*, 51(2):98–104, 1981.

[24] E. Goles-Chacc, F. Fogelman-Soulie, and D. Pellegrin. Decreasing energy functions as a tool for studying threshold networks. *Discret. Appl. Math.*, 12(3):261–277, 1985.

[25] Y. Hassin and D. Peleg. Distributed probabilistic polling and applications to proportionate agreement. *Inform. Comput.*, 171(2):248 – 268, 2001.

[26] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proc. Natl. Acad. Sci. USA*, 79:2554–2558, 1987.

[27] S. Huang. Gene expression profiling, genetic networks, and cellular states: an integrating concept for tumorigenesis and drug discovery. *J. Mol. Med.*, 77(6):469–480, 1999.

[28] D. Kempe, J. Kleinberg, and E. Tardos. Maximizing the spread of influence through a social network. In *Proc. of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '03, pages 137–146, New York, NY, USA, 2003. ACM.

[29] L. B. Kier, P. G. Seybold, and C. K. Cheng. *Cellular Automata Modeling of Chemical Systems.* Cellular Automata Modeling of Chemical Systems: A Textbook and Laboratory Manual. Springer, 2005.

[30] J. Knutson. A survey of the use of cellular automata and cellular automata-like models for simulating a population of biological cells. Graduate theses and dissertations, Iowa State University, USA, 2011.

[31] S. Kutten and D. Peleg. Fault-local distributed mending. *J. Algorithms*, 30(1):144 – 165, 1999.

[32] A. R. Mikler, S. Venkatachalam, and K. Abbas. Modeling infectious diseases using global stochastic cellular automata. *J. Biol. Syst.*, 13:421–439, 2005.

[33] A. Montanari and A. Saberi. Convergence to equilibrium in local interaction games. *SIGecom Exch.*, 8(1):11:1–11:4, July 2009.

[34] S. Morris. Contagion. *Rev. Econ. Stud.*, 67:57–78, 2000.

[35] N. H. Mustafa and A. Pekeč. Listen to your neighbors: How (not) to reach a consensus. *SIAM J. Discret. Math.*, 17(4):634–660, 2004.

[36] L. I. L. Oliveira, V. C. Barbosa, and F. Protti. The predecessor-existence problem for k-reversible processes. *Theor. Comput. Sci.*, 562:406–418, 2015.

[37] P. Orponen. Computational complexity of neural networks: A survey. *Nordic J. of Computing*, 1(1):94–110, 1994.

[38] C. H. Papadimitriou. *Computational Complexity.* Addison Wesley Pub. Co., 1994.

[39] D. Peleg. Local majorities, coalitions and monopolies in graphs: a review. *Theor. Comput. Sci.*, 282(2):231 – 257, 2002. {FUN} with Algorithms.

[40] J. M. Perdang and A. Lejeune. *Cellular Automata: Prospects in Astrophysical Applications.* World Scientific, 1993.

[41] S. Poljak and M. Sůra. On periodical behaviour in societies with symmetric influences. *Combinatorica*, 3(1):119–121, 1983.

[42] S. Poljak and D. Turzík. On an application of convexity to discrete systems. *Discret. Appl. Math.*, 13(1):27 – 32, 1986.

[43] S. Poljak and D. Turzík. On pre-periods of discrete influence systems. *Discret. Appl. Math.*, 13(1):33 – 39, 1986.

[44] G. Y. Vichniac. Simulating physics with cellular automata. *Phys. D: Nonlinear Phenom.*, 10(1):96 – 116, 1984.

[45] J. Šíma and P. Orponen. General-purpose computation with neural networks: A survey of complexity theoretic results. *Neural Comput.*, 15(12):2727–2778, 2003.

Appendix C:

And/Or-Convexity: A Graph Convexity Based on Processes and Deadlock Models

This appendix contains the article "*A Graph Convexity Based on Processes and Deadlock Models*" submitted on March 2015 to *Annals of Operations Research* journal.

Annals of Operations Research manuscript No.
(will be inserted by the editor)

And/Or-Convexity: A Graph Convexity Based on Processes and Deadlock Models

Carlos V.G.C. Lima · Fábio Protti · Dieter Rautenbach · Uéverton S. Souza · Jayme L. Szwarcfiter

Received: date / Accepted: date

Abstract Deadlock prevention techniques are essential in the design of robust distributed systems. However, despite the large number of different algorithmic approaches to detect and solve deadlock situations, yet there remains quite a wide field to be explored in the study of deadlock-related combinatorial properties. In this work we consider a simplified AND-OR model, where the processes and their communication are given as a graph G. Each vertex of G is labelled AND or OR, in such a way that an AND-vertex (resp., OR-vertex) depends on the computation of all (resp., at least one) of its neighbors. We define a graph convexity based on this model, such that a set $S \subseteq V(G)$ is convex if and only if every AND-vertex (resp., OR-vertex) $v \in V(G) \setminus S$ has at least one (resp., all) of its neighbors in $V(G) \setminus S$. We relate some classical convexity parameters to blocking sets that cause deadlock. In particular, we show that those parameters in a graph represent the sizes of minimum or maximum blocking sets, and also the computation time until system stability is reached. Finally, a study on the complexity of combinatorial problems related to such graph convexity is provided.

Keywords Graph Convexity · Deadlock · AND-OR Model · And/Or Graphs

1 Introduction

Let V denote a set of processes in a distributed computation. Informally, as described by Barbosa and Benevides (1998), a deadlock is said to exist in this computation if a subset $S \subseteq V$ can be identified whose members are all blocked due to the occurrence of some condition that can only be relieved by members of the same subset S. In other words, a deadlock situation is characterized by the permanent impossibility for a group of processes to progress with

Carlos V.G.C. Lima and Jayme L. Szwarcfiter
PESC, COPPE, Federal University of Rio de Janeiro, Rio de Janeiro, Brazil
E-mail: {gclima,jayme}@cos.ufrj.br

Fábio Protti and Uéverton S. Souza
Institute of Computing, Fluminense Federal University, Niterói, Brazil
E-mail: {fabio,ueverton}@ic.uff.br

Dieter Rautenbach
Institute of Optimization and Operations Research, Ulm University, Ulm, Germany
E-mail: dieter.rautenbach@uni-ulm.de

their tasks due to the occurrence of a condition that prevents at least one needed resource from being granted to each of the processes in that group. Note that a *necessary* condition for the existence of a deadlock in the distributed computation is the existence of cycles of dependency.

A useful abstraction to analyze deadlock situations is the wait-for graph $G = (V,E)$, where E is a set of directed edges, and an edge exists in E directed away from $v_i \in V$ towards $v_j \in V$ if v_i is blocked for some condition that v_j may relieve. The graph G changes dynamically as the computation progresses, and what determines the evolution of G by allowing for changes in the set of its directed edges is the deadlock model that holds for the computation (Barbosa and Benevides, 1998). In essence, what a deadlock model does is to specify rules for vertices that are not *sinks* in G to become sinks. (A sink is a vertex with out-degree zero).

In this paper we are concerned with *stable properties*. Once a stable property takes hold of a group of processes, only an external intervention that eventually follows its detection can break it. Whenever we refer to G, we mean the wait-for graph that corresponds to a "snapshot" of the distributed computation in the usual sense of a consistent global state (Barbosa, 1996; Chandy and Lamport, 1985).

In this work we show that combinatorial concepts from graph convexity have a close relation to stable properties of a wait-for graph in the AND-OR model. In essence, such properties can be regarded as structures associated with graph convexity invariants whose definitions are based on the wait-conditions in the AND-OR model. Some examples are the size of a maximum knot (b-knot (Barbosa and Benevides, 1998)), or the minimum number of processes that must become sinks in order to eliminate deadlock. Our studies consider convexity properties of wait-for graphs, directed graphs, and their undirected underlying graphs. The concepts of graph convexity and AND-OR deadlock model will be detailed later.

We summarize the results of this paper as follows:

1. We prove that the AND-OR model defines a graph convexity in both cases of directed and undirected graphs. In such a convexity, the graphs are defined in such a way that each vertex depends on all (AND-vertex) or on at least one (OR-vertex) of its neighbors to be realized. Moreover, we present applications of some convexity parameters to stable properties related to the AND-OR model;
2. Having defined the convexity, we study the complexity of determining some of its convexity parameters. The first one is the *convexity number*, and we prove that computing this parameter can be done in linear-time;
3. On the other hand, we prove that computing the *interval number* of such a convexity is an *NP*-hard problem. In particular, we prove that it is true even if all the vertices are AND-vertices, as well as all vertices are OR-vertices. Furthermore, we also prove that it is *NP*-hard even if there exists only one AND-vertex and the graph is bipartite. We also present an efficient $(2+\log|V_{\text{or}}|)$-approximation algorithm to compute the interval number, where V_{or} denotes the set of all OR-vertices. Finally, we present a linear-time algorithm to compute the interval number for trees, where we adapt the algorithm of Cockayne, Goodman, and Hedetniemi (1975);
4. We also deal with the problem of determining the *hull number*. We prove that there is always a minimum *hull set* containing no OR-vertices. In particular, we show that a minimum hull set can be described as a minimum vertex cover of all the AND-vertices, which covers all the OR-vertices. We also prove an efficient $(2+\log|V_{\text{or}}|)$-approximation algorithm to compute the hull number. Moreover, we show how to adapt the linear-time

algorithm that determines the interval number for trees, such that it computes the hull number for trees, keeping the same complexity;
5. Finally we prove that the *Carathéodory number* can be determined in linear-time.

2 Additional Concepts and Notations

In this section we provide the necessary definitions and notations used in the remainder of the text. We first explain the AND-OR model. Next, we give the related definitions from graph convexities. Moreover we show how the convexity parameters can be related to stable properties of the AND-OR model. Finally we prove that AND/OR graphs define a graph convexity.

2.1 The AND-OR Model

In this work we use standard terminology and notation of finite graphs (Bondy and Murty, 2008) with n vertices and m edges. Let G be a directed graph with vertex set $V(G) = \{v_1, v_2, \ldots, v_n\}$. For $v_i \in V(G)$, let $N^+_G(v_i)$ denote the set of *descendants* of v_i in G (nodes that are reachable from v_i, including itself) and $N^-_G(v_i)$ denote the set of *ancestors* of v_i in G (nodes from which v_i is reachable, including itself). Let $D_i \subseteq N^+_G(v_i)$ be the set of immediate descendants of $v_i \in G$ (descendants that are one edge away from v_i) and $A_i \subseteq N^-_G(v_i)$ its set of immediate ancestors in G (ancestors that are one edge away from v_i). Nodes in $N^+_G(v_i) \setminus N^-_G(v_i)$ are called *subordinates* of v_i.

As in (Barbosa and Benevides, 1998), a *deadlock model* for the distributed computation underlying G is a collection of subsets $W_i^1, \ldots, W_i^{p_i}$ of D_i for all $v_i \in V(G)$, where:

- $W_i^1 \cup \cdots \cup W_i^{p_i} = D_i$;
- No two nonempty sets in $W_i^1, \ldots, W_i^{p_i}$ are such that one is a subset of the other;
- In order to exit its blocked state and proceed with its local computation, a node v_i for which $D_i \neq \emptyset$ must receive a signal from all nodes in at least one of the nonempty sets $W_i^1, \ldots, W_i^{p_i}$.

At this level of generality, the deadlock model is known as the *AND-OR model*, reflecting the need for v_i to be signaled by all members of W_i^1 (if nonempty), or all members of W_i^2 (if nonempty), and so on. If at most one of $W_i^1, \ldots, W_i^{p_i}$ is nonempty for all $v_i \in V(G)$, then the deadlock model is the *AND model*. Similarly, if all nonempty sets in $W_i^1, \ldots, W_i^{p_i}$ are singletons for all $v_i \in V(G)$, then the deadlock model is known as the *OR model*. A sufficient condition for the existence of a deadlock in the AND model is the same as the general necessary condition mentioned earlier, that is, the existence of a directed cycle in G. For the OR model, a necessary and sufficient condition is the existence of a *knot* in G. A knot is a subset $K \subseteq V(G)$ with $|K| > 1$ such that $N^+_G(v_i) = K$, for all $v_i \in K$. For details on these conditions and related material, the reader is referred to (Knapp, 1987; Singhal, 1989) and the references therein.

Situations that can be characterized by the AND-OR model are, for example, those in which v_i perceives several conjunctions of resources as equivalent to one another, and issues requests for several of them with provisions to withdraw some of them later (Barbosa and Benevides, 1998; Brzezinski et al., 1995; Ryang and Park, 1995; Kshemkalyani and Singhal, 1994).

We consider in our analysis a simplified AND-OR model in which there are two types of processes, AND and OR. An AND process v_i can only become a sink when its wait is relieved by all the processes in D_i, whereas an OR process v_i can be released by any positive number of processes in D_i. Simplified AND-OR model generalizes AND model and OR model, and although it is a natural subcase of AND-OR model, it is easy to see that a system in the general AND-OR model can be transformed in polynomial-time into a system operating according to the simplified AND-OR model.

For short, we simply say *and/or graph* to refer to a wait-for graph in the simplified AND-OR model. And/or graphs are a well-known data structure in the literature, with many applications in different fields of Computer Science. For instance, artificial intelligence (Nilsson, 1971; Simon and Lee, 1971), game theory, robotics, operational research, automation and software engineering. And/or graphs model cutting problems (dos S. Souza, 2014; Morabito and Pureza, 2010), robotic task plans (Cao and Sanderson, 1998), assembly/disassembly sequences (de Mello and Sanderson, 1991), evaluation of boolean formulas (Laber, 2008), failure dependencies (Barnett and Verma, 1994), software versioning (Conradi and Westfechtel, 1998), game trees (Kumar and Kanal, 1984), composition of web services (Ma, Dong, and He, 2008) and distributed systems (Barbosa, 2002). The optimization problem of minimizing the solution subgraph of an edge-weighted and/or graph was considered in (dos S. Souza, Protti, and da Silva, 2013; Sahni, 1974), where complexity aspects from both the classical and parameterized point of view are dealt with.

2.2 The And/Or-Convexity

As stated earlier, we link the study of and/or graphs with *convexity on graphs*, a discipline that has received broad attention recently, specially for the *geodetic*, *monophonic*, and P_3 convexities (Araujo et al., 2013; Cáceres et al., 2006; Campos et al., 2015; Coelho et al., 2014; da F. Ramos et al., 2014; Dourado et al., 2009, 2010, 2012).

Formally, given a graph G, a *graph convexity* is a pair $(G,\mathcal{C})$ such that:

- $\mathcal{C}$ is a collection of subsets of $V(G)$ closed under intersections;
- $\emptyset \in \mathcal{C}$;
- $V(G) \in \mathcal{C}$.

Every $C \in \mathcal{C}$ is called *convex*. We define a family $\mathcal{C}^*$ of vertex subsets of an undirected underlying and/or graph G as follows:

$(*)$ A set C is a member of $\mathcal{C}^*$ if and only if every or-vertex in $V(G)\setminus C$ does not have neighbors in C and every and-vertex in $V(G)\setminus C$ has at least one neighbor in $V(G)\setminus C$.

Analogously, for the directed version, we define a family $\mathcal{C}^{**}$ of vertex subsets of an and/or graph G as follows:

$(**)$ A set C is a member of $\mathcal{C}^{**}$ if and only if every or-vertex in $V(G)\setminus C$ does not have *out-neighbors* in C and every and-vertex in $V(G)\setminus C$ has at least one *out-neighbor* in $V(G)\setminus C$.

Now, we prove that $(G,\mathcal{C}^*)$ and $(G,\mathcal{C}^{**})$ are graph convexities, called *and/or-convexities* on undirected and directed graphs, respectively. The main goal of this work is to relate the convexity parameters defined below with stable properties of a *distributed computation in the AND-OR model* that uses an and/or graph as the network topology.

Theorem 1 $(G, \mathcal{C}^*)$ *defines a graph convexity.*

Proof By $(*)$, it is clear that $\emptyset \in \mathcal{C}^*$ and $V(G) \in \mathcal{C}^*$. To prove that $\mathcal{C}^*$ is closed under intersections, consider two sets in $\mathcal{C}^*$, A and B, and prove that $C = A \cap B$ is in $\mathcal{C}^*$. Let $v \notin A$. If v is an or-vertex, then v has no neighbor in A, and thus in C. If v is an and-vertex, then $N_G(v) \nsubseteq A$, and thus $N_G(v) \nsubseteq C$. By symmetry on the vertices no in B, it follows that $C \in \mathcal{C}^*$. □

Corollary 2 $(G, \mathcal{C}^{**})$ *defines a graph convexity.*

Proof The proof is similar to Theorem 1. □

2.3 Some Convexity Parameters and Their Applications

Now, we present some convexity invariants used in this paper. Furthermore, for each invariant we present interpretations with respect to stable properties of a distributed computation.

The ***convex hull*** of a subset $S \subseteq V(G)$ is the smallest convex set $H(S)$ that contains S. If $H(S) = V(G)$, then S is a ***hull set*** of G. The cardinality $hn(G)$ of a minimum hull set of G is the ***hull number*** of G.

– *Application:* A convex set C can be viewed as a set of processes with the following property: relieving all the processes in C from their wait condition, with the corresponding dispatch of grant messages, cannot relieve any process $w \notin C$ from its wait condition. In addition, for every subset of processes S, the minimum convex set C containing S is the set of all processes that can be transitively relieved by those of S. Hence, a hull set S is a minimum set of processes such that all processes in $V(G) \setminus S$ can be transitively relieved by those of S.

It follows by $(*)$ that $V(G)$ and the empty set are convex sets, where we say that they are *trivial convex sets*. The ***convexity number*** of G is the largest cardinality $cx(G)$ of a non-trivial convex set $C \subset V(G)$.

– *Application:* A b-knot is a structure in G that can be used to characterize deadlocks under the AND-OR model (Barbosa and Benevides, 1998). As can be observed in Subsection 3.1, the convexity number of a directed graph G and the size of a minimum b-knot of G, in fact, are complementary quantities. Hence, computing $cx(G)$ is equivalent to calculating the size of a minimum b-knot of G, which is $n - cx(G)$.

The ***Carathéodory number*** is the smallest integer $c(G)$ such that for every $S \subseteq V(G)$ and every $u \in H(S)$, there is a subset $X \subseteq S$ in which $|X| \leq c(G)$ and $u \in H(X)$.

– *Application:* Any process can be transitively relieved by an appropriated subset of processes of G. The Carathéodory number of G, $c(G)$, means that each process p of G can be transitively relieved by a subset of processes X_p with $|X_p| \leq c(G)$.

We say that a graph convexity $(G, \mathcal{C})$ is an *interval convexity* when it admits a function $I: 2^{V(G)} \to 2^{V(G)}$, called *interval function*, where for every $S \subseteq V(G)$, $I(S) = S \cup W$, where W is a set of vertices determined by some properties of $\mathcal{C}$. If $I(S) = S$, then S is convex. The most common graph convexities, as the geodetic, monophonic, and P_3 convexities, are all interval convexities, each one having its corresponding interval function. Similarly, we define an interval function for and/or-graphs as follows. Let $I_0[S] = S$ and $I_t[S] = I_{t-1}[S] \cup W_t$ for any positive integer t, where W_t is the set of all or-vertices (resp. and-vertices) with at least one neighbor (resp. all neighbors) in $I_{t-1}[S]$.

The ***interval number*** of G is the minimum cardinality $in(G)$ of a subset $S \subseteq V(G)$ satisfying $I_1[S] = V(G)$. We call such a set S as an ***interval set*** of G.

– *Application:* Computing the interval number and the hull number of undirected graphs are special cases of such problems on directed graphs (an undirected edge can be regarded as a pair of directed edges, one for each direction). In practice, the interval number expresses the smallest set of processes that must send grant messages at the initial step of the computation such that all the remaining processes are relieved from their wait condition in exactly one time step. On the other hand, the hull number expresses the minimum number of processes that must send grant messages at the initial step of the computation such that all remaining processes are eventually relieved from their wait condition.

The remainder of the text is organized as follows. In Section 3 we prove the following results on the and/or convexities defined above:

– In Subsection 3.1 we show that computing the convexity number can be done in linear time for every and/or graph;
– In Subsection 3.2 we prove that deciding if the interval number of an and/or graph G is equal to a given integer k is an NP-complete problem even if G has only and- or only or-vertices. In addition, we show that it remains NP-complete even if G is a bipartite graph having only one and-vertex. We also present a $(2+\log|V(G_{\text{or}})|)$ approximation algorithm to compute $in(G)$, where $V(G_{\text{or}})$ denotes the set of all or-vertices of G. Moreover we present a linear-time algorithm to compute the interval number for trees.
– In Subsection 3.3 we prove that there is always a minimum hull set containing no or-vertices if G contains and-vertices. In particular, contracting each induced or-components of G to one vertex, we show that a minimum hull set can be described as a minimum vertex cover of all the and-vertices and such that it covers all the or-vertices. Moreover, we show how to adapt the $(2+\log|V(G_{\text{or}})|)$ approximation algorithm and the linear-time algorithm for the interval number for general graphs and trees, respectively, so that they compute their hull number.
– Finally, in Subsection 3.4 we prove that the Carathéodory number can be obtained in linear time for every and/or graph.

In Section 4 we present our conclusions.

3 Computing the Convexity Parameters

In this section, we prove a number of results on the classical convexity parameters defined previously. Let us suppose only connected graphs in this paper. We need some additional definitions. An *and/or graph* G is a graph whose all vertices are labelled as one of $\{\text{and,or}\}$. Let G_{or} (resp. G_{and}) denote the induced subgraph defined by all the or-vertices (resp. and-vertices) of G. Moreover, let G^i_{or} be the i-th-connected component of G_{or}, $1 \leq i \leq \omega(G_{\text{or}})$, such that $\omega(G_{\text{or}})$ denotes the number of connected components of G_{or}. The *closed neighborhood* of $S \in V(G)$ is defined as $N[S] = S \cup N_G(S)$, where $N_G(X) = \bigcup_{v\in X} N_G(v)$. We will refer to G always as an and/or graph in the remainder of the text.

3.1 The Convexity Number

In this subsection we prove that there exists a linear time algorithm which computes $cx(G)$ for every undirected graph G. Let $\theta(G) = \min_{1\leq i\leq \omega(G_{\text{or}})} |N_G[V(G^i_{\text{or}})]|$ denote the smallest cardinality of the closed neighborhood among all the or-connected components of G.

Theorem 3 *Given an undirected connected and/or-graph G, it follows that*

$$cx(G) = \begin{cases} 0 & , \textit{if } V(G_{\text{and}}) = \emptyset; \\ n-2 & , \textit{if } V(G_{\text{and}}) \neq \emptyset \textit{ and } V(G_{\text{and}}) \textit{ is not an independent set}; \\ n-\theta(G) & , \textit{otherwise}. \end{cases}$$

Furthermore $cx(G)$ can be computed in linear time.

Proof If G does not contain any and-vertex, then $I_t[\{v\}] = V(G)$ for every vertex v and some time step t. Thus the convex sets of G are only the trivial convex sets and thus $cx(G) = 0$. Hence, suppose that there must exist at least one and-vertex in $V(G)$. For every vertex v, the subset $V(G) \setminus \{v\}$ is not convex, implying that $cx(G) \leq n-2$. If there exists an edge (u,v) such that u and v are and-vertices, then $V(G) \setminus \{u,v\}$ is a convex set, since u and v are neighbors outside $V(G) \setminus \{u,v\}$. Thus, we get that $cx(G) = n-2$.

Let us suppose now that $V(G_{\text{and}})$ induces an independent set. Note that a non-trivial convex set cannot contain all the or-vertices of G. Moreover, if an or-vertex v does not belong to a non-trivial convex set C then no vertex of the connected component G^i_{or} containing v could be in C either. Furthermore, if an and-vertex $v \in N_G(V(G^i_{\text{or}}))$ belongs to C, then C must include the entire or-component G^i_{or}. Hence $N_G[V(G^i_{\text{or}})]$ must be in $V(G) \setminus C$ and the maximum cardinality of C is equal to $n - \theta(G)$. It is not hard to see that computing $\theta(G)$ can be done in $\Theta(n+m)$ time. □

Corollary 4 *The convexity number of directed connected graphs can be computed in polynomial time.*

Proof Let G be a directed graph. If G has a sink vertex v, then $V(G) \setminus \{v\}$ is a maximum non-trivial convex set. Otherwise, if G has only and-vertices, then $V(G) \setminus C_k$ is a maximum non-trivial convex set, where C_k is a minimum directed cycle of G. Such a cycle can be found in polynomial time by adapting the Floyd-Warshall algorithm (Cormen et al., 2009). If G has both and- and or- vertices, then $V(G) \setminus B$ is a maximum non-trivial convex set, where B is a minimum b-knot of G, which can also be found in polynomial time (Barbosa and Benevides, 1998). □

3.2 The Interval Number

Computing the interval number of an and/or graph G is NP-hard in general. If all vertices are labelled "or" then determining $in(G)$ is equivalent to finding the cardinality of a minimum *dominating set* of G, denoted by $\gamma(G)$, which is an NP-hard problem even for split or bipartite graphs (Bertossi, 1984). On the other hand, if all vertices are labelled "and" then computing $in(G)$ is equivalent to determining the cardinality of a minimum *vertex cover* of G, denoted by $\beta(G)$. This is true because, for each pair of adjacent vertices u and v that are not in a subset $S \subset V(G)$, they cannot be in $I_t[S]$, for any $t > 0$. Thus, $V(G) \setminus S$ must be an independent set in G, for any interval set S, which implies that S is a vertex cover of G. Since it is NP-hard to compute $\beta(G)$ even for cubic planar graphs (Garey and Johnson, 1990), computing $in(G)$ is also NP-hard.

Therefore, every interval set S must contain a vertex cover of G_{and} and every vertex in $V(G) \setminus S$ has a neighbor in S. Moreover, every minimum vertex cover of G is an interval set of G. From these observations we get that

$$\min\{\gamma(G), \beta(G_{\text{and}})\} \leq in(G) \leq \beta(G).$$

Next, we consider the complexity of determining $in(G)$ for general bipartite graphs. We consider the following decision problem:

INTERVAL NUMBER
INSTANCE: An and/or graph G and an integer $k \geqslant 1$.
QUESTION: Is $in(G) = k$?

Theorem 5 INTERVAL NUMBER *is NP-complete for connected bipartite graphs with exactly one* and*-vertex.*

Proof We consider a reduction from the VERTEX COVER problem for cubic graphs with at least six vertices, which remains NP-complete (Garey and Johnson, 1990). Given a cubic graph $G = (V,E)$, we construct a bipartite graph $G' = (V',E')$ such that

$$V(G') = \{w_{v_i} : v_i \in V(G)\} \cup \{w_{v_iv_j} : v_iv_j \in E(G)\} \cup \{w\}$$

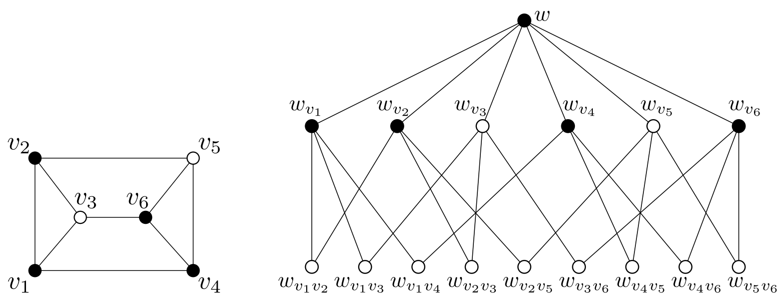
and

$$E(G') = \{ww_{v_i} : w_{v_i} \in V(G')\} \cup \{w_{v_i}w_{v_lv_r} : i = l \text{ or } i = r, w_{v_i} \in V(G') \text{ and } w_{v_lv_r} \in V(G')\}.$$

Finally, assign “and” to w and “or” to the remaining vertices of $V'(G')$. Fig. 1 shows an example of the construction. Observe that G' is a bipartite graph with $1+5n/2$ vertices and $4n$ edges. Its parts are defined by $\{w\} \cup \bigcup_{v_iv_j \in E(G)} \{w_{v_iv_j}\}$ and $\bigcup_{v_i \in V} \{w_{v_i}\}$.

Now we will prove that $in(G') = \beta(G)+1$. Notice that the set $S = \{w\} \cup C$ is an interval set of G', where C is a minimum vertex cover of G, since every vertex $w_{v_iv_j}$ has a neighbor $w_{v_i} \in C$ and every vertex w_{v_i} is adjacent to w. Thus $in(G') \leq |S| = \beta(G)+1$.

Let S be a minimum interval set of G'. If $w \notin S$ then every vertex w_{v_i} must be in S, implying that $in(G) = n$, since S is an interval set of G'. Therefore $n \leq \beta(G)+1$, which implies that G is a clique. In other words, $G = K_4$, a contradiction by the assumption $n \geq 6$. Hence $w \in S$ and we can obtain a new interval set S' from S as follows. For every $w_{v_\ell v_r} \in S$, if both w_{v_ℓ} and w_{v_r} are not in S, replace $w_{v_\ell v_r}$ by either w_{v_ℓ} or w_{v_r}. Otherwise, just remove $w_{v_\ell v_r}$ of S. We can see that S' is an interval set of G' smaller than S. Moreover S' is also a minimum vertex cover of G. Hence $\beta(G)+1 = |S'| \leq |S| = in(G')$, completing the proof. $\square$



(a) A cubic graph G. (b) Bipartite graph G' obtained from G.

Fig. 1: Example of transformation from graph G to G' in Theorem 5. The black vertices are a minimum vertex cover of G in Fig. 1a and a minimum interval set of G' in Fig. 1b.

Corollary 6 *Determining the interval number of a directed connected graph G is NP-hard.*

Proof Given an undirected and/or graph G, we can construct a directed and/or graph G' replacing each edge $e = (u,v)$ by two arcs $a_e^1 = (u,v)$ and $a_e^2 = (v,u)$. Assuming that G has only and-vertices, it is easy to see that G has a vertex cover of size k if and only if G' has an interval set of size k. Note that, if G has only or-vertices we obtain a reduction from the dominating set problem. □

3.2.1 Approximation Algorithm

Algorithm 1 describes an approximation procedure for the interval number. Given a connected and/or graph G, a set S is an interval set if and only if

- $G \setminus S$ does not contain any edge uv, where u or v is an and-vertex, and
- every or-vertex in $V(G) \setminus S$ has a neighbor in S.

The first condition implies that if G_1 is the spanning subgraph of G that contains all edges uv of G where u or v is an and-vertex, then $in(G) \geq \beta(G_1)$. The second condition implies $in(G) \geq \gamma(G, V(G_{\text{OR}}))$, where $\gamma(G, V(G_{\text{OR}}))$ is the minimum cardinality of a set D of vertices of G such that every vertex in $V(G_{\text{OR}}) \setminus D$ has a neighbor in D. The union of a 2-approximate vertex cover of G_1 and of a $\log|V(G_{\text{OR}})|$-approximate $V(G_{\text{OR}})$-dominating set of G yields a $(2+\log|V(G_{\text{OR}})|)$-approximate interval set of G.

In fact, one can easily construct an equivalent set cover instance in order to carry over to $\gamma(G, V(G_{\text{OR}}))$, where the ground set to be covered is the set of or-vertices, and the sets are the closed neighborhoods of all vertices of G.

Algorithm 1: A $(2+\log|V(G_{\text{OR}})|)$-approximation algorithm for $in(G)$.

```
Data: A connected and/or Graph $G$.
Result: An interval set $S$ of $G$ of size at most $(2+\log|G_{\text{OR}}|) * in(G)$.
1 $G_1 \leftarrow$ A spanning subgraph of $G$ that contains all edges $uv$ of $G$ where $u$ or $v$ is an and-vertex;
2 $S \leftarrow$ The matched vertices of a maximal matching in $G_1$;
3 $U \leftarrow V(G_{\text{OR}})$;
4 while $U \neq \emptyset$ do
5     $v \leftarrow$ The vertex of $G$ whose $N_G[v]$ contains the largest number of uncovered or-vertices;
6     Cover all vertices in $N[v] \cap V(G_{\text{OR}})$;
7     $U \leftarrow U \setminus N[v]$;
8     $S \leftarrow S \cup \{v\}$;
Result: $S$
```

Theorem 7 *Given a connected and/or graph G, Algorithm 1 is a* $(2+\log|V(G_{\text{OR}})|)$*-approximation algorithm for* $in(G)$*. Furthermore computing* $in(G)$ *is a Log-APX-complete problem.*

Proof Let us denote the size of an optimum solution by OPT and the size returned by Algorithm 1 for an instance I by $A(I)$. We know that every interval set of G must include a vertex cover of G_{AND}. Thus, line 1 represents the classical 2-approximation algorithm to obtain a vertex cover (Vazirani, 2001). After the line 2 it is obtained a set S_1 that covers all edges between and-vertices. Since any interval set must be a vertex cover for G_{and}, it follows that

$$|S_1|/2 \leq OPT \implies |S_1| \leq 2 * OPT. \quad (1)$$

Lines 3–8 present the $\log n$-approximation algorithm for DOMINATING SET (Vazirani, 2001), where U represents the universal set and the sets are composed by $|N_G[v]|$, for every $v \in V(G)$. Hence, all or-vertices are covered by a set S_2, following that

$$|S_2| \leq \log(|V(G_{\text{OR}})|) * OPT. \tag{2}$$

From Eq. (1) and Eq. (2) we obtain the following for any instance I:

$$\begin{aligned} A(I) &= |S_1| + |S_2| \\ &\leq 2 * OPT + \log(|V(G_{\text{OR}})|) * OPT \\ &= (2 + \log|V(G_{\text{OR}})|) * OPT. \end{aligned}$$

Since DOMINATING SET is in Log-APX-complete (Escoffier and Paschos, 2006), the theorem follows. □

Corollary 8 *Algorithm 1 is a $(1+\log|V(G_{\text{OR}})|)$-approximation for $in(G)$, if G is a bipartite connected and/or graph. Moreover, if G is such that one part has only* and*-vertices and the other has only* or*-vertices, then $in(G)$ can be computed in polynomial time.*

Proof By the well-known König-Egerváry's theorem, a minimum vertex cover for G can be found in polynomial time for bipartite graphs. Thus the first step in Algorithm 1 results in a vertex cover for the and-vertices of size at most OPT. Hence the approximation is limited by the dominating step of the or-vertices, obtaining an approximation of $(1+\log|V(G_{\text{OR}})|)$.

When G has each part formed by vertices of the same type, distinct from the type of the other part, we can see that every interval set S must be a minimum vertex cover of G. This is true because no edges can exist between vertices of $V(G) \setminus S$. Hence the proof follows by the König-Egerváry's theorem. □

3.2.2 Linear Time Algorithm for Trees

Now we prove that INTERVAL NUMBER is polynomially solvable for trees. We adapt the classical linear time algorithm presented in (Cockayne, Goodman, and Hedetniemi, 1975), which computes $\gamma(T)$ for a tree T. In fact, the algorithm works for a more general domination called *mixed domination*, defined as follows. Consider a partition P of the vertices of G into three subsets V_1, V_2, V_3 containing *free*, *bound*, and *required* vertices, respectively. Such a partition P is called an *m-partition*. A *mixed dominating set D* of a graph *G with respect to P* (or simply *wrt P*) is a subset of vertices containing all required vertices and such that every bound vertex either belongs to D or is adjacent to some vertex of D. Furthermore, we require an additional restriction on each and-vertex v: $v \notin D$ if and only if $N_G(v) \subseteq D$. We denote the size of a minimum mixed dominating set of G wrt P by $\gamma_m(G,P)$.

Proposition 9 *There exists an m-partition P of a connected graph G such that $in(G) = \gamma_m(G,P)$.*

Proof Let S be a minimum interval set of G. This implies that every or-vertex of $V(G) \setminus S$ has at least one neighbor in S and all and-vertices in $V(G) \setminus S$ have their entire neighborhood in S. By setting $V_1 = V_3 = \emptyset$ and $V_2 = V(G)$ we obtain an m-partition P such that S is a mixed dominating set D of G wrt P. This shows that $\gamma_m(G,P) \leq in(G)$. Now, let D be a minimum mixed dominating set of G wrt P. Clearly, every and-vertex v of $V(G) \setminus D$ has all of its neighbors in D, and every or-vertex of $V(G) \setminus D$ is covered by at least one vertex of D. Hence D is also an interval set of G, which implies $in(G) \leq \gamma_m(G,P)$. □

The next theorem is the basis of the algorithm that computes $\gamma_m(T,P)$ for a tree T. We assume without loss of generality that all leaves of T are or-vertices. Denote by $G-v$ the graph obtained by removing v from G. Moreover, for each and-vertex v, let $r(v)$ denote the number of its required neighbors.

Theorem 10 *Let T be a tree with an m-partition P, v a leaf of T, u the neighbor of v, and $T' = T - v$. Let P_v be the m-partition of T' obtained by removing v from the member of P that contains v. Then:*

1. *if $v \in V_1$ and $u \in V(T_{\text{and}})$ then $\gamma_m(T,P) = \gamma_m(T',P')$, where P' is the m-partition obtained from P_v by setting u as required;*
2. *if $v \in V_2$ then $\gamma_m(T,P) = \gamma_m(T',P')$, where P' is the m-partition obtained from P_v by setting u as required;*
3. *if $v \in V_3$ and $u \in V_3$ then $\gamma_m(T,P) = 1 + \gamma_m(T',P_v)$;*
4. *if $v \in V_3$ and $u \notin V_3$ then $\gamma_m(T,P) = 1 + \gamma_m(T',P'')$, where P'' is the m-partition of T' obtained by setting u as free if either $u \in V(T_{\text{or}})$ or $u \in V(T_{\text{and}})$ and $r(u) = |N_T(u) \cap V(T)|$.*

Proof (1) Since $v \in V_1$, any mixed dominating set of T' wrt P' is also a mixed dominating set of T wrt P, and therefore $\gamma_m(T,P) \le \gamma_m(T',P')$. To prove that $\gamma_m(T',P') \le \gamma_m(T,P)$, let D be a minimum mixed dominating set of T wrt P. If $v \notin D$ then D is also a mixed dominating set of T' wrt P'. Thus, suppose that $v \in D$; this implies $u \notin D$ (otherwise $D \setminus \{v\}$ is a mixed dominating set of T wrt P, contradicting the optimality of D). Let $D' = (D \setminus \{v\}) \cup \{u\}$. If $u \in V(T_{\text{and}})$ then D' is a mixed dominating set of T' wrt P', because $v \notin (N_T(u) \cap D')$. Hence $\gamma_m(T',P') \le |D'| = |D| = \gamma_m(T,P)$.

(2) Since u is required in T', every mixed dominating set of T' wrt P' is also a mixed dominating set of T wrt P, implying that $\gamma_m(T,P) \le \gamma_m(T',P')$. The proof of inequality $\gamma_m(T',P') \le \gamma_m(T,P)$ follows as in item (1).

(3) The proof of this case is trivial.

(4) Let D' be a mixed dominating set of T' wrt P''. From the definition of P'', it is clear that $D \cup \{u\}$ is a mixed dominating set of T wrt P. Therefore $\gamma_m(T,P) \le 1 + \gamma_m(T',P'')$. Now, let D be a minimum mixed dominating set of T wrt P. Since v is required, $v \in D$. Moreover, if $u \in D$ then $D \setminus \{v\}$ is a mixed dominating set of T' wrt P''. Finally, assume $u \notin D$. Since u is free in T', we have two cases: (a) if u is an or-vertex then u is dominated for some vertex $w \in D \setminus \{u,v\}$; (b) if u is an and-vertex then $r(u) = |N_T(u) \cap V(T)|$. In both cases it is clear that $D \setminus \{v\}$ is a mixed dominating set of T' wrt P''. Hence, $\gamma_m(T',P'') \le |D| - 1 = \gamma_m(T,P) - 1$. □

Theorem 10 directly leads to the following linear time procedure to compute a minimum mixed domination set for a labelled tree:

In the above algorithm, $r(v)$ is initially set to zero for each and-vertex v. Lines 7–10 cover cases (1) and (2) in Theorem 10, whereas lines 11–15 cover cases (3) and (4). If $v \in V_1$ then v may or may not belong to a mixed dominating set of T. Thus, if v is an or-vertex then it is either a leaf of T or has a required neighbor already processed. On the other hand, if v is an and-vertex then $r(v) = d(v)$, and therefore v is the last vertex in the computation. If $r(v) = d(v) - 1$ then v is a bound vertex, since v depends on whether $u \in N_T(v)$ is in D or not, where u has not been analyzed yet.

Algorithm 2: Linear time algorithm that computes $\gamma_m(T,P)$ for a tree T.

```
   Data: An and/or tree $T$ whose vertices are labelled as free, bound, or required.
   Result: A minimum mixed dominating set MDS of $T$.
1  MDS $\leftarrow \emptyset$;
2  for every $v \in V(T_{\text{and}})$ do
3      $r(v) \leftarrow 0$;
4  repeat
5      $v \leftarrow$ a leaf of $T$;
6      $u \leftarrow$ neighbor of $v$ in $T$;
7      if (($v$ is free and $u \in V(T_{\text{and}})$) or $v$ is bound ) and $u$ is not required then
8          set $u$ as required;
9          for every $w \in N_T(u) \cap V(T_{\text{and}})$ do
10             $r(w) \leftarrow r(w) + 1$;
11     else
12         if $v$ is required then
13             MDS $\leftarrow$ MDS $\cup \{v\}$;
14             if $u$ is not required and ($u \in V(T_{\text{or}})$ or ($u \in V(T_{\text{and}})$ and $r(u) = |N_T(u) \cap V(T)|$)) then
15                 set $u$ as free;
16     $T \leftarrow T - v$;
17 until $n = 1$;
18 if the last vertex $v$ is not free then
19     MDS $\leftarrow$ MDS $\cup \{v\}$;
```

3.3 The Hull Number

In this section we present some complexity and structural properties on hull sets of and/or graphs G. Computing $hn(G)$ is trivially solvable if G contains only or-vertices – in this case every vertex defines itself a hull set. However, as described in Subsection. 3.2, if G contains no or-vertex then determining $hn(G)$ is an NP-hard problem, since $\beta(G_{\text{and}}) \leq hn(G) \leq in(G)$ and so $hn(G) = \beta(G) = \beta(G_{\text{and}})$. Based on this fact, we obtain the following result:

Lemma 11 *Let G be a connected graph containing* and-*vertices. For any hull set S of G, there exists a hull set S' with no* or-*vertices and such that $|S'| \leq |S|$.*

Proof Let S be a hull set of G and let $v \in S$ be an or-vertex. If $N_G(v) \cap S \setminus \{v\} \neq \emptyset$, then $S' = S \setminus \{v\}$ is also a hull set of G. Otherwise, if there exists at least one and-vertex $u \in N_G(v) \cap (V(G) \setminus S)$, then $S' = \{u\} \cup S \setminus \{v\}$ is also a hull set of G, since $S \subset I_1[S']$. Finally, assume that $N_G(v)$ is entirely contained in G^i_{or}. Thus, if $N_G(V(G^i_{\text{or}}) \setminus \{v\}) \cap S \neq \emptyset$, then $S' = S \setminus \{v\}$ is a hull set of G. Otherwise, all the and-neighbors of G^i_{or} have at least one neighbor outside S. Hence let $S' = \{u\} \cup S \setminus \{v\}$, where u is such an and-neighbor of $N_G(V(G^i_{\text{or}}) \setminus \{v\})$. It is easy to see that S' is a hull set of G. Applying the same reasoning to each or-vertex $v \in S$ we obtain a new hull set as required in the statement.

Let G^* be the and/or graph obtained from G contracting each or-component G^i_{or} to a unique vertex v^i, for all $1 \leq i \leq \omega(G_{\text{or}})$. In other words, replace $V\left(G^i_{\text{or}}\right)$ by v^i such that $N\left(v^i\right) = N\left(V(G^i_{\text{or}})\right) \cap V\left(G_{\text{and}}\right)$. We say that G^* is the *contracted graph* of G. Moreover, for each subset $S \subseteq V(G)$ let us define the *contracted set* $S^* \subseteq V(G^*)$ of S as follows:

- $v \in S \cap V(G_{\text{and}})$ if and only if $v \in S^* \cap V(G_{\text{and}})$;
- if $v \in S \cap V(G^i_{\text{or}})$ then $v^i \in S^*$;
- if $v^i \in S^*$ then there exists at least one vertex $u \in S \cap V(G^i_{\text{or}})$.

Observe that the or-vertices of G^* induce an independent set.

Lemma 12 *Let G^* be the contracted graph of G. For every $S \subseteq V(G)$, $H(S) \cap V(G_{\text{and}}) = H(S^*) \cap V(G_{\text{and}})$.*

Proof If S does not contain any or-vertex, then $S^* = S$ and the lemma is trivial. Since every and-vertex of S is in S^*, and vice versa, let $v \in H(S) \cap (V(G_{\text{and}}) \setminus S)$. Thus $N_G(v) \subseteq I_t[S]$ for some $t \in \mathbb{N}$, and suppose by contradiction that $v \notin H(S^*)$. For every $u \in N_G(v) \cap V(G^i_{\text{or}})$, we get that $u \in I_{t^*}[S]$ for some $t^* \leq t$, implying that $S \cap N[V(G^i_{\text{or}})] \neq \emptyset$. Hence $S^* \cap N[v^i] \neq \emptyset$ and thus $v^i \in H[S^*]$. Therefore, there must exist an and-vertex $w \in N_G(v)$ such that $w \in H(S)$ and $w \notin H(S^*)$. If $w \notin S$ then w and v depend on each other to be in $H(S)$, which implies that $w \notin H(S)$. This means that $w \in S$ and thus $w \in S^*$. Hence $w \in H(S^*)$, a contradiction.

Conversely, consider $v \in H(S^*) \cap (V(G_{\text{and}}) \setminus S^*)$. As observed before, each and-neighbor of v must be in S^* and thus in S. Moreover, each $v^i \in N_G(v)$ belongs to $I_t[S^*]$ for some $t \in \mathbb{N}$. Thus there must exist an or-vertex $u \in V(G^i_{\text{or}})$ such that $u \in H(S)$. Hence $V(G^i_{\text{or}}) \subset H[S]$, i.e., $N_G(v) \subseteq H(S)$ in G. □

Theorem 13 *For every connected and/or graph G, $hn(G) = hn(G^*)$.*

Proof Let S be a minimum hull set of G that contains only and-vertices, which exists by Lemma 11. Since all the or-vertices of G must be in $H(S)$, it follows that every or-component of G must be "reached" by S. Thus all the or-vertices of G^* are also reached by S. By Lemma 12, S is also a hull set of G^*, implying that $hn(G^*) \leq hn(G)$.

Conversely, let S^* be a minimum hull set of G^* containing only and-vertices. Since all the or-vertices of G^* belong to $H(S^*)$, at least one vertex of each or-connected component of G^* either is in S or is adjacent to one vertex of S. Hence all the or-vertices of G belong to $H(S^*)$. Therefore, by Lemma 12, S^* is also a hull set of G, that is, $hn(G) \leq hn(G^*)$. □

By Theorem 13, we assume that G is contracted in the remaining of this subsection.

Corollary 14 *If $V(G_{\text{and}})$ is an independent set, then computing $hn(G)$ is NP-hard.*

Proof In this case G is a bipartite graph whose parts are formed by the and-vertices and the or-vertices, respectively. Since every or-vertex must be in $H(S)$ for every hull set S, we get that all and-vertices must also be in $H(S)$. Thus, it follows by Lemma 11, Lemma 12 and, Theorem 13 that $hn(G)$ is the size of a minimum set S of and-vertices which covers all the or-vertices. In other words, computing a minimum hull number is equivalent to determining a minimum SET COVER of G, where the or-vertices are the elements and each and-vertex v represents the subset of elements that v is adjacent. Since computing a minimum SET COVER is *NP*-hard, the corollary follows. □

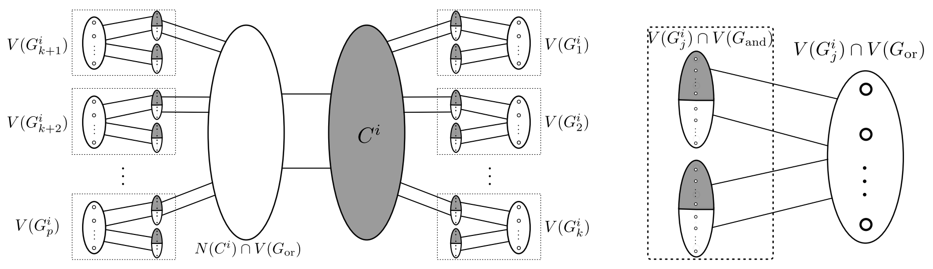
Now, let us consider that the and-vertices does not induce an independent set. Let C^i be the i-th vertex cover of G_{and}. Moreover, let G^i be the subgraph induced by the subset $V(G) \setminus \left(C^i \cup \left(N\left(C^i\right) \cap V\left(G_{\text{or}}\right)\right)\right)$, and let G^i_j denote the j-th connected component of G^i. We use the notation $H' \Subset H$ to mean that H' is a connected component of graph H. Fig. 2a shows a representation of a hull set of G by the gray sets.

Theorem 15 *The hull number of a connected graph G can be determined as*

$$hn(G) = \min_{C^i} \left\{ |C^i| + \sum_{G^i_j \Subset G^i} hn\left(G^i_j\right) \right\}.$$

Proof Let C^i be a vertex cover of G_{and}, and let S denote the set formed by C^i and the union of the minimum hull sets of each G^i_j, that contain only and-vertices by Lemma 11. Let us prove that $hn(G) \leq \min\left\{ |C^i| + \sum_{G^i_j \subseteq G^i} hn\left(G^i_j\right) \right\}$. Note that every or-vertex has at least one neighbor in S, otherwise their and-neighbors cannot be in $H(S)$, a contradiction. Furthermore, since $V(G_{\text{and}}) \setminus S$ is an independent set, we get that all the and-vertices are reached in at most two time steps. Hence S is a hull set of G.

Conversely, let S be a minimum hull set of G which contains only and-vertices. We know that S must contain a vertex cover C^i of G_{and}, otherwise the endpoints of an edge between and-vertices are neighbors outside S. Therefore, every connected G^i_j component of G^i is a bipartite graph with parts containing only and-vertices and only or-vertices, respectively. Hence, a minimum hull set of G containing only and-vertices can be defined by taking a vertex cover of G_{and} that minimizes the sum of the hull numbers of the generated connected components. Therefore $hn(G) \geq \min\left\{ |C^i| + \sum_{G^i_j \subseteq G^i} hn\left(G^i_j\right) \right\}$. □


$V(G^i_{k+1})$
$V(G^i_{k+2})$
$V(G^i_p)$
$N(C^i) \cap V(G_{\text{or}})$
C^i
$V(G^i_1)$
$V(G^i_2)$
$V(G^i_k)$
$V(G^i_j) \cap V(G_{\text{and}})$
$V(G^i_j) \cap V(G_{\text{or}})$

(a) Hull set of G. (b) Component G^i_j.

Fig. 2: Representation of a hull set of G given by the gray sets.

We can see a minimum hull set of G as a smallest set S of and-vertices that covers all the or-vertices and such that $V(G_{\text{and}}) \setminus S$ is an independent set. Furthermore, a hull set S containing only and-vertices is also an interval set of G if and only if S covers all the or-vertices and every and-vertex outside S has their whole neighborhood in S.

Corollary 16 *Algorithm 1 is a $(2 + \log|V(G_{\text{OR}})|)$-approximation algorithm for $hn(G)$. Furthermore computing $hn(G)$ is a Log-APX-complete problem.*

Proof The proof is analogous to that of Theorem 7. □

Corollary 17 *Determining the hull number of a directed connected graph G is NP-hard.*

Proof Assuming that G has only and-vertices the interval and hull numbers are equal. □

3.3.1 Linear Time Algorithm for Trees

Given a tree T, we can adapt Algorithm 2 to find its hull number, where $V(T)$ is partitioned into free, bound and required vertices. However, the definition of mixed domination is slightly different in this case. Formally, a mixed dominating set D^* of T is as follows: D^* contains all required vertices; every bound vertex belongs to D^* or is covered by it; $v \notin D^*$ if and only if $N_G(v) \cap V(G_{\text{and}}) \subseteq D^*$, for every and-vertex v. We denote by $\gamma_m^*(T,P)$ the size of a minimum mixed dominating set of T wrt an m-partition P into free, bound, and required vertices. The following proposition is the analogous of Proposition 9 for $\gamma_m^*(G,P)$.

Proposition 18 *Let G be a connected and/or graph. Then there exists an m-partition P of G such that $hn(G) = \gamma_m^*(G,P)$.*

Proof Let S be a minimum hull set of G containing no or-vertices. Then S must cover all the or-vertices of G, and every and-vertex of $V(G) \setminus S$ must have its entire neighborhood in S. Thus S is a mixed dominating set of G wrt the m-partition $P = (\emptyset, V(G), \emptyset)$, and this implies $\gamma_m^*(G,P) \leq hn(G)$. Now, let D^* be a minimum mixed dominating set of G wrt P. Thus $N_G(v) \cap V(G_{\text{and}}) \subseteq D^*$ if and only if $v \notin D^*$, for each and-vertex v. Let w be an or-vertex such that $w \in D^*$. If w is a bound vertex and $N_G(w) \cap D^* \neq \emptyset$ then $D^* \setminus \{w\}$ is also a mixed dominating set of G wrt P, a contradiction. Hence there must exist $u \in N_G(w) \cap (V(G) \setminus D^*)$, and then $(D^* \setminus \{w\}) \cup \{u\}$ is also a minimum mixed dominating set of G wrt P. However, if w is a required vertex, we can set w as bound and apply the same arguments, for each such or-vertex. The set D' obtained in this way contains only and-vertices and every or-vertex is adjacent to some vertex of D'. Hence D' is a hull set of G, implying that $hn(G) \leq |D'| = |D^*| = \gamma_m^*(G,P)$. □

Let us denote by $r^*(v)$ the number of required and-neighbors of v, for every and-vertex v. The following theorem describes how to compute a minimum hull set on trees.

Theorem 19 *Let T be a tree with an m-partition P, v a leaf of T, u the neighbor of v, and $T' = T - v$. Let P_v be the m-partition of T' obtained by removing v from the member of P that contains v. Then:*

1. *if $v \in V_1$ then $\gamma_m^*(T,P) = \gamma_m^*(T',P_v)$;*
2. *if $v \in V_2$ then $\gamma_m^*(T,P) = \gamma_m^*(T',P')$, where P' is the m-partition obtained from P_v by setting u as required;*
3. *if $v \in V_3$ and $u \in V_3$ then $\gamma_m^*(T,P) = 1 + \gamma_m^*(T',P_v)$;*
4. *if $v \in V_3$ and $u \notin V_3$ then $\gamma_m^*(T,P) = 1 + \gamma_m^*(T',P'')$, where P'' is the m-partition of T' obtained by setting u as free if either $u \in V(T_{\text{or}})$ or $u \in V(T_{\text{and}})$ and $r^*(u) = |N_T(u) \cap V(T_{\text{and}})|$.*

Proof (1) Since $v \in V_1$, any mixed dominating set of T' wrt P_v is a mixed dominating set of T wrt P, and thus $\gamma_m^*(T,P) \leq \gamma_m^*(T',P_v)$. Let D^* be a minimum mixed dominating set of T wrt P. If $v \notin D^*$ then D^* is also a mixed dominating set of T' wrt P_v. Suppose then that $v \in D^*$. If v is an and-vertex then $r^*(v) = |N_T(v) \cap V(G_{\text{and}})|$. Moreover, if $u \in V(G_{\text{and}})$ then u must be labelled as required, which implies that u also belongs to D^*. Hence, if $u \in D^*$ then $D^* \setminus \{v\}$ is also a mixed dominating set of T wrt P even if v is either an and- or an or-vertex, a contradiction. Hence we get that $u \notin D^*$, and then $D' = (D^* \setminus \{v\}) \cup \{u\}$ is also a mixed dominating set of T wrt P. We conclude that $\gamma_m^*(T',P_v) \leq |D'| = |D^*| = \gamma_m^*(T,P)$.

(2) Since v is dominated by u in every mixed dominating set D^* of T' wrt P', D^* is also a mixed dominating set of T wrt P, implying that $\gamma_m^*(T,P) \leq \gamma_m^*(T',P')$. Now, let D^* be a minimum mixed dominating set of T wrt P. If $v \notin D^*$ then D^* is also a mixed dominating set of T' wrt P', and this means that u must be in D^*. Thus u is a required vertex and D^* is also a mixed dominating set of T' wrt P'. Then suppose that $v \in D^*$. If $u \in D^*$ then $D^* \setminus \{v\}$ is a mixed dominating set of T wrt P, a contradiction. Then let us suppose that $u \notin D^*$. Consider

$D' = (D^* \setminus \{v\}) \cup \{u\}$. It is easy to see that D' is a mixed dominating set of T' wrt P'. Hence $\gamma_m^*(T',P') \leq |D'| = |D^*| = \gamma_m^*(T,P)$.

(3) The proof is trivial.

(4) The proof of this case is analogous to case (4) in Theorem 10, where we show that every mixed dominating set of T' wrt P'' plus u is a mixed dominating set of T wrt P. Furthermore, we prove that u is either an or-vertex and has a neighbor in T' or u is an and-vertex with $r^*(u) = |N_T(v) \cap V(T_{\text{and}})|$. □

Algorithm 3 presents a linear time procedure that computes the minimum mixed dominating set $\gamma^*(T,P)$ of an and/or tree T.

Algorithm 3: Linear time algorithm that computes $\gamma^*(T,P)$ for a tree T.

```
Data: An and/or tree $T$ whose vertices are labelled as free, bound or required.
Result: A minimum mixed dominating set MDS$^*$ of $T$.
MDS$^* \leftarrow \emptyset$;
for every $v \in V(T_{\text{and}})$ do
    $r^*(v) \leftarrow 0$;
repeat
    $v \leftarrow$ a leaf of $T$;
    $u \leftarrow$ neighbor of $v$ in $T$;
    if $v$ is bound and $u$ is not required then
        set $u$ as required;
        for every $w \in N_T(u) \cap V(T_{\text{and}})$ do
            $r^*(w) \leftarrow r^*(w) + 1$;
    else
        if $v$ is required then
            MDS$^* \leftarrow$ MDS$^* \cup \{v\}$;
            if $u$ is not required and $(u \in V(T_{\text{or}})$ or $(u \in V(T_{\text{and}})$ and $r^*(u) = |N_T(u) \cap V(T_{\text{and}})|))$
            then
                set $u$ as free;
    $T \leftarrow T - v$;
until $n = 1$;
if the last vertex $v$ is not free then
    MDS$^* \leftarrow$ MDS$^* \cup \{v\}$;
```

3.4 The Carathéodory Number

Now, we present results on the Carathéodory number in the and/or-convexity.

Lemma 20 *Given a connected and/or graph G, it follows that*

$$c(G) = \begin{cases} 1 & \text{, if } G \text{ contains only or-vertices;} \\ \Delta(G) & \text{, if } G \text{ contains only and-vertices.} \end{cases} \tag{3}$$

Proof Since each vertex defines a hull set of G, if G contains only or-vertices, the theorem trivially holds in this case. Assume that G contains only and-vertices. The convex hull of any subset $S \subset V(G)$ must contain the entire neighborhood of each vertex in $H(S) \setminus S$. This implies that $c(G) \leq \Delta(G)$. Let $S = N_G(v)$, where $d(v) = \Delta(G)$. The convex hull of every subset $X \subset S$ cannot contain v, although $v \in H(S)$. Hence $c(G) \geq \Delta(G)$. □

Lemma 20 implies that $c(G) \geq \max\{1, \Delta(G_{\text{and}})\}$, where $\Delta(G_{\text{and}})$ denotes the maximum degree of G_{and}. Next we present a polynomial-time algorithm that computes $c(G)$ for every graph G. Let $d_{\text{and}}(v)$ denote the number of and-neighbors of v. In addition, given an and-vertex v, let $\overline{d_{\text{or}}}(v)$ denote the number of or-neighbors of v that are not adjacent to any and-vertex in the neighborhood of v. Formally, $\overline{d_{\text{or}}}(v) = \left| N_{\text{or}}(v) \setminus \bigcup_{u \in N_{\text{and}}(v)} N_G(u) \right|$.

Theorem 21 *Given a connected and/or graph G, it follows that*

$$c(G) = \max\{1, \max_{v \in V(G_{\text{and}})} \{d_{\text{and}}(v) + \overline{d_{\text{or}}}(v)\}\}.$$

Furthermore, computing $c(G)$ can be done in linear time.

Proof By Lemma 20, if G contains only or-vertices then $c(G) = 1$ and the theorem trivially holds. Hence, let $S \subseteq V(G)$ containing both and-vertices and or-vertices. Since the or-vertices in $H(S)$ can be obtained by taking all subsets $X \subseteq S$ with $|X| = 1$, we can consider only the and-vertices in $H(S) \setminus S$. Thus, let v be such an and-vertex. As in the proof of Lemma 20, every and-neighbor of v must be in any subset $X \subset S$ such that $v \in H(X)$. Also, the entire neighborhood of v should satisfy this condition. Thus, if there exists an or-neighbor w of v which is not adjacent to any and-neighbor of v, w must be included in X. Hence $|X| \geq d_{\text{and}}(v) + \overline{d_{\text{or}}}(v)$ and it is clear to see that $c(G) \geq \max_{v \in V(G_{\text{and}})}\{d_{\text{and}}(v) + \overline{d_{\text{or}}}(v)\}$. Furthermore, it is not hard to see that $c(G) \leq \max_{v \in V(G_{\text{and}})}\{d_{\text{and}}(v) + \overline{d_{\text{or}}}(v)\}$, otherwise there must exist an and-vertex that is not in $H(S)$, which contradicts the maximality of $d_{\text{and}}(v) + \overline{d_{\text{or}}}(v)$. Finally, it is not hard to see that computing $c(G)$ can be done in $\Theta(n+m)$ time, which concludes the proof. □

4 Conclusion

In this work we related some special graph convexity parameters to combinatorial properties of distributed computation and deadlocks. Up to the authors' knowledge, no study on Distributed Systems has been done under this point of view. We showed that some structures of blocking sets in the AND-OR model are actually well-known optimization problems on graphs, such as the dominating set problem and the vertex cover problem.

Furthermore, such parameters describe the trade-off between the cost of the number of processes initially active and the total time of the computation, as exemplified by the interval number and the hull number. Although in both concepts a minimum vertex set is required for the computation to reach all the nodes, the former demands a quick execution, while the latter is not concerned about the time – and this allows choosing a smaller set at the beginning of the computation. For example, in a deadlocked distributed system, where all processes are blocked, a minimum hull set is equivalent to the minimum number of processes required to be released in order to ensure that all the processes will eventually be reached by the computation. Since deadlock is a stable property, removing it from a distributed computation requires some external intervention. Hence, a minimum hull set is precisely a minimum set of processes to which some external operations must be applied in order to guarantee that all the processes will eventually be relieved from their wait condition.

We leave open the study of other convexity parameters, such as the Radon number, the rank, and the Helly number. What is the complexity of computing them in the AND-OR model? What are their meanings in this model?

Acknowledgements Partially supported by Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), Fundação de Amparo à Pesquisa do Estado do Rio de Janeiro (FAPERJ) and Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) – Brazilian research agencies.

References

Araujo J, Campos V, Giroire F, Sampaio L, Soares R (2013) On the hull number of some graph classes. Theor Comput Sci 475:1–12

Barbosa VC (1996) An Introduction to Distributed Algorithms. MIT Press

Barbosa VC (2002) The combinatorics of resource sharing. In: Corrêa R, Dutra I, Fiallos M, Gomes F (eds) Models for Parallel and Distributed Computation, Appl. Opt., vol 67, Springer US, pp 27–52

Barbosa VC, Benevides MRF (1998) A graph-theoretic characterization of and-or deadlocks. Tech. rep., Federal University of Rio de Janeiro

Barnett J, Verma T (1994) Intelligent reliability analysis. In: Proceedings of the Tenth Conference on Artificial Intelligence for Applications, 1994., IEEE, pp 428–433

Bertossi AA (1984) Dominating sets for split and bipartite graphs. Inform Process Lett 19(1):37–40

Bondy JA, Murty US (2008) Graph theory, volume 244 of Graduate Texts in Mathematics. Spring-Verlag Press

Brzezinski J, Helary JM, Raynal M, Singhal M (1995) Deadlock models and a general algorithm for distributed deadlock detection. J Parallel Distr Com 31(2):112–125

Cáceres J, Hernando C, Mora M, Pelayo IM, Puertas ML, Seara C (2006) On geodetic sets formed by boundary vertices. Discrete Math 306(2):188–198

Campos V, Sampaio RM, Silva A, Szwarcfiter JL (2015) Graphs with few P_4s under the convexity of paths of order three. Discrete Appl Math 192(0):28–39

Cao T, Sanderson AC (1998) And/or net representation for robotic task sequence planning. IEEE T Syst Man CY C 28(2):204–218

Chandy KM, Lamport L (1985) Distributed snapshots: determining global states of distributed systems. ACM T Comput Syst 3(1):63–75

Cockayne E, Goodman S, Hedetniemi S (1975) A linear algorithm for the domination number of a tree. Inform Process Lett 4(2):41–44

Coelho EM, Dourado MC, Rautenbach D, Szwarcfiter JL (2014) The Carathéodory number of the convexity of chordal graphs. Discrete Appl Math 172(0):104–108

Conradi R, Westfechtel B (1998) Version models for software configuration management. ACM Comput Surv 30(2):232–282

Cormen TH, Leiserson CE, Rivest RL, Stein C (2009) Introduction to Algorithms, Third Edition, 3rd edn. The MIT Press

Dourado MC, Gimbel JG, Kratochvíl J, Protti F, Szwarcfiter JL (2009) On the computation of the hull number of a graph. Disc Math 309(18):5668 – 5674

Dourado MC, Protti F, Szwarcfiter JL (2010) Complexity results related to monophonic convexity. Discrete Appl Math 158(12):1268–1274

Dourado MC, Rautenbach D, dos Santos VF, Szwarcfiter JL (2012) Characterization and recognition of Radon-independent sets in split graphs. Inform Process Lett 112(24):948–952

Escoffier B, Paschos VT (2006) Completeness in approximation classes beyond $\{$APX$\}$. Theor Comput Sci 359(1–3):369 – 377 da F Ramos I, dos Santos VF, Szwarcfiter JL (2014) Complexity aspects of the computation of the rank of a graph. Discrete Math & Theor Comput Sci 16(2):73–86

Garey MR, Johnson DS (1990) Computers and Intractability; A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., New York, NY, USA

Knapp E (1987) Deadlock detection in distributed databases. ACM Comput Surv 19(4):303–328

Kshemkalyani AD, Singhal M (1994) Efficient detection and resolution of generalized distributed deadlocks. IEEE T Software Eng 20(1):43–54

Kumar V, Kanal LN (1984) Parallel branch-and-bound formulations for and/or tree search. IEEE T on Pattern Anal 6(6):768–778

Laber ES (2008) A randomized competitive algorithm for evaluating priced and/or trees. Theor Comput Sci 401(13):120–130

Ma X, Dong B, He M (2008) And/or tree search algorithm in web service composition. In: Pacific-Asia Workshop on Computational Intelligence and Industrial Application, 2008. PACIIA ’08., vol 2, pp 23–27

de Mello LSH, Sanderson AC (1991) A correct and complete algorithm for the generation of mechanical assembly sequences. IEEE T Robotic Autom 7(2):228–240

Morabito R, Pureza V (2010) A heuristic approach based on dynamic programming and and/or-graph search for the constrained two-dimensional guillotine cutting problem. Ann Oper Res 179(1):297–315

Nilsson NJ (1971) Problem-Solving Methods in Artificial Intelligence. McGraw-Hill Pub. Co.

Ryang DS, Park KH (1995) A two-level distributed detection algorithm of and/or deadlocks. J Parallel Distr Com 28(2):149–161

dos S Souza U (2014) Multivariate Investigation of NP-Hard Problems: Boundaries Between Parameterized Tractability and Intractability. Graduate theses, Fluminense Federal University, Brazil

dos S Souza U, Protti F, da Silva MD (2013) Revisiting the complexity of and/or graph solution. J Compu Syst Sci 79(7):1156–1163

Sahni S (1974) Computationally related problems. SIAM J Comput 3(4):262–279

Simon R, Lee RCT (1971) On the optimal solutions to and/or series-parallel graphs. J ACM 18(3):354–372

Singhal M (1989) Deadlock detection in distributed systems. Computer 22(11):37–48

Vazirani VV (2001) Approximation Algorithms. Springer-Verlag New York, Inc., New York, NY, USA

Appendix D:

Generalized Threshold Processes on Graphs

This appendix contains the article "*Generalized Threshold Processes on Graphs*" submitted on January 2015 to *Theoretical Computer Science* journal.

Generalized Threshold Processes on Graphs

Carlos V.G.C. Lima[1] Dieter Rautenbach[2] Uéverton S. Souza[3] Jayme L. Szwarcfiter[1]

[1] PESC, COPPE, Federal University of Rio de Janeiro
Rio de Janeiro, Brazil, {gclima,jayme}@cos.ufrj.br

[2] Institute of Optimization and Operations Research, Ulm University
Ulm, Germany, dieter.rautenbach@uni-ulm.de

[3] Institute of Computing, Fluminense Federal University
Niterói, Brazil, ueverton@ic.uff.br

Abstract

We consider an iterative irreversible process on graphs. Starting with some initial set S of vertices of a given graph G, this process iteratively adds to S all vertices u of G outside of S for which the intersection of the current set S with the neighborhood $N_G(u)$ of u in G belongs to a collection $\tau(u)$ of subsets of $N_G(u)$ given for each vertex u. Based on discrete convexity notions, we study the corresponding interval number, where only one iteration is executed, and present some results on the hull number, where the number of iterations is unbounded. Special choices of the function τ allow to include several well studied graph processes and parameters within this framework. The results of the paper are the following: the general problems are NP-hard in several very restricted cases, that is, even on extremely simple graphs sufficiently complex threshold functions lead to hard problems; in view of the strong hardness results, linear time algorithms for trees in the τ interval version and for paths in the τ hull version are given; a probabilistic upper bound on the size of a solution for the relaxed τ-interval number.

Keywords: Irreversible Threshold Processes · Target Set · Dynamic Monopoly · Domination

1 Introduction

Motivated by completely different applications such as distributed computing, cellular automata, marketing strategies, spreading of infectious diseases, discrete convexity notions, and physical percolation effects, several iterative irreversible processes on graphs have been studied in recent years [1–5, 8, 13, 16, 17, 24, 26]. Such processes typically consider a set of vertices of a given graph that grows iteratively according to certain extension rules. These rules usually involve threshold values, which may be vertex-dependent. A specific vertex might for instance enter some set of 'infected' vertices provided that sufficiently many of its neighbors are already contained in that set. In the present paper we consider a generalization of such processes, where we replace the threshold values by specific sets, that is, each vertex may individually specify certain subsets of its neighborhood, and the process is guided by these specific selections.

Before giving the precise definitions, we recall some classical terminology and notation.

We consider finite, simple, and undirected graphs. The vertex set and the edge set of a graph G are denoted by $V(G)$ and $E(G)$, respectively. For a vertex u of a graph G, the neighborhood $N_G(u)$ of u in G is the set $\{v \in V(G) : uv \in E(G)\}$, the degree $d_G(u)$ of u in G is $|N_G(u)|$, and the closed neighborhood $N_G[u]$ of u in G is $\{u\} \cup N_G(u)$. The minimum and maximum degree of a graph G are denoted by $\delta(G)$ and $\Delta(G)$, respectively.

For an integer k and a set V, let $[k]$ be the set of all positive integers at most k, let 2^V be the set of all subsets of V, and let $\binom{V}{k}$ be the set of all subsets of V of order k.

Based on discrete convexity notions [6,18], now we give formal definitions for the generalized threshold processes.

Let G be a graph. A *threshold function* on G is a function $\tau : V(G) \to 2^{V(G)}$ such that $\tau(u) \subseteq 2^{N_G(u)}$ for every vertex u of G.

Let S be a set of vertices of G. The *τ-interval* $I_\tau(S)$ of S is the set

$$S \cup \{u : u \in V(G) \setminus S \text{ and } N_G(u) \cap S \in \tau(u)\},$$

that is, $I_\tau(S)$ contains S as well as all vertices u from the complement of S whose neighborhood intersects S in a set that is contained in $\tau(u)$.

Let $I^0_\tau(S) = S$, and, for every positive integer i, let $I^i_\tau(S) = I_\tau(I^{i-1}_\tau(S))$. If k is the smallest non-negative integer with $I^{k+1}_\tau(S) = I^k_\tau(S)$, then $I^k_\tau(S)$ is the *τ-hull* $H_\tau(S)$ of S, and the sequence $\left(I^0_\tau(S), \ldots, I^k_\tau(S)\right)$ is the *τ-threshold process on G starting with S.*

If $I_\tau(S) = S$, then S is *τ-convex.* If $I_\tau(S) = V(G)$, then S is a *τ-interval set.* Finally, if $H_\tau(S) = V(G)$, then S is a *τ-hull set.* The minimum order of a τ-interval set is the *τ-interval number* $i_\tau(G)$ of G, and the minimum order of a τ-hull set is the *τ-hull number* $h_\tau(G)$ of G.

A motivation for the study of τ-threshold processes on graphs is that they generalize some gates of logical circuits. For example, an `and`-gate v outputs *true* only if all of its inputs are *true*; on the other hand, for a `xor`-gate outputs *true* it must receive an odd number of positive inputs. A vertex with $\tau(v) = N(v)$ operates according to an `and`-gate and when $\tau(v) = \{N \in 2^{N_G(u)} : |N| \mod 2 = 1\}$ the vertex operates as an `xor`-gate.

We also consider a relaxed version of the above notions, where a vertex u from $V(G) \setminus S$ belongs to the interval of S if $N_G(u) \cap S$ *contains* some set from $\tau(u)$ instead of requiring to be equal to some such a set. In other words, a way of introducing this variant is by suitably modifying the function τ. Therefore, let the *closure* $\bar{\tau}$ of τ be the threshold function

$$\bar{\tau} : V(G) \to 2^{V(G)} : u \mapsto \left\{\bar{N} : \bar{N} \subseteq N_G(u) \text{ and } \exists N \in \tau(u) : N \subseteq \bar{N}\right\}.$$

The sets $I_{\bar{\tau}}(S)$ and $H_{\bar{\tau}}(S)$ are the *relaxed τ-interval of S* and the *relaxed τ-hull of S*, respectively. The $\bar{\tau}$-threshold process on G starting with S is the *relaxed τ-threshold process on G starting with S.* The *relaxed τ-interval number* of G is $i_{\bar{\tau}}(G)$, and the *relaxed τ-hull number* of G is $h_{\bar{\tau}}(G)$.

Special choices for τ lead to many well known graph parameters.

If $\tau(u) = 2^{N_G(u)} \setminus \{\emptyset\}$ for every vertex u of some graph G, then $i_\tau(G)$ coincides with the *domination number* $\gamma(G)$ [22]. Alternatively, if $\tau(u) = \binom{N_G(u)}{1}$, then $i_{\bar{\tau}}(G)$ also coincides with the domination number $\gamma(G)$. More generally, if $\tau(u) = \binom{N_G(u)}{k}$ for some positive integer k, then $i_{\bar{\tau}}(G)$ coincides with the *k-domination number* $\gamma_k(G)$ [9,13,19,22,25,27], and the relaxed τ-threshold processes correspond to the processes considered in [1,5,8,12,13,16,26].

The above examples already imply several hardness results.

In view of algorithmic aspects though, some remarks concerning encoding lengths are necessary. Given a graph G of order $n(G)$, and a threshold function τ on G, we can encode the pair (G, τ) listing, for every vertex u of G, the $d_G(u)$ neighbors of u by indicating their names, which are 0/1-vectors of length $O(\ln n(G))$, as well as the $|\tau(u)|$ distinct 0/1-incidence vectors of length $d_G(u)$ of the sets in $\tau(u)$, where the i-th entry of such a vector corresponds to the neighbor of u with the i-th smallest name. This encoding has size

$$O\left(\sum_{u \in V(G)} \Big(d_G(u)\Big(\ln(n(G)) + |\tau(u)|\Big)\Big)\right). \tag{1}$$

In the remainder of this work we assume the input is encoded as above. Note that $|\tau(u)|$ can be $\Omega\left(2^{d_G(u)}\right)$, hence this size is not necessarily polynomial in $n(G)$. However, we assume that the hardest instances have size $O(poly(n(G)))$.

An interesting threshold function based on `xor`-gates of logic circuits is the following parity related function:

$$\oplus : V(G) \to 2^{V(G)} : u \mapsto \left\{N \in 2^{N_G(u)} : |N| \mod 2 = 1\right\}.$$

Clearly, the reasonable encoding length of the pair $(G, \oplus)$ is just the encoding length of G, which is much smaller than the expression in (1). Note that $\oplus$-hull number of a tree equals 1.

The results of the paper can be summarized as follows:

1. Results about the τ-interval number and τ-interval sets:
   - NP-completeness of computing the τ-interval number on complete graphs (Theorem 3).
   - NP-completeness of computing the τ-interval number for very special threshold functions (Theorems 4 and 5); Theorem 5 is about computing the so-called $\oplus$-interval number on bipartite graphs.
   - A linear time algorithm for computing the *tau*-interval number of a tree (Theorem 12).
   - A linear time algorithm for determining whether a given tree has a τ-interval set that does not contain a given vertex (Proposition 10).
2. Results about the relaxed τ-interval number:
   - NP-completeness and inapproximability results for computing the relaxed τ-interval number on complete graphs (Theorems 1 and 2).
   - A linear time algorithm for computing the relaxed τ-interval number for a given tree (Corollary 13).
   - A linear time algorithm for computing the $\oplus$-interval number of a tree (Theorem 14).
   - A probabilistic upper bound is given for the relaxed τ-interval parameter (Theorem 15, Corollary 16), together with an example suggesting that there is no reasonable version of Corollary 16 for the usual τ-interval number.
3. Results for the τ-hull number:
   - A polynomial time algorithm for computing the τ-hull number of a path (Proposition 8).
4. Results for the relaxed τ-hull number:
   - NP-completeness and inapproximability results for computing the relaxed τ-hull number on complete graphs (Theorems 1 and 2).
   - A linear time algorithm for computing the relaxed *tau*-hull number of a tree (Theorem 7).

2 Results

We organize our results in three subsections.

2.1 Hardness results

Our first hardness results concern complete graphs, that is, even on extremely simple graphs sufficiently complex threshold functions lead to hard problems.

Theorem 1 *Given a pair (k, τ), where k is a non-negative integer and τ is a threshold function on a complete graph K_n, it is NP-complete to decide whether $i_{\bar{\tau}}(K_n) \leq k$ and it is NP-complete to decide whether $h_{\bar{\tau}}(K_n) \leq k$.*

Proof: The two considered decision problems are clearly in NP, because the $\bar{\tau}$-threshold process on G starting with a given set can be generated in polynomial time. In order to establish hardness, we describe polynomial reductions from Set Cover. Therefore, let $\mathcal{C} = (k, (S_1, \ldots, S_p))$ be an instance of Set Cover, that is, k is a positive integer, the S_i are subsets of some ground set V, and $\mathcal{C}$ is a 'Yes'-instance of Set Cover if and only if there is a set C of at most k integers in $[p]$ with $\bigcup_{i \in C} S_i = V$. Clearly, we may assume that $\bigcup_{i \in [p]} S_i = V$ and $k < \min\{p, |V|\}$.

Let $n = p + |V|$, and let $V(K_n) = \{S_1, \ldots, S_p\} \cup V$. For $i \in [p]$, let $\tau(S_i) = \{\emptyset\}$, and, for $u \in V$, let $\tau(u) = \{\{S_i\} : i \in [p] \text{ and } u \in S_i\}$. It is easy to see that $\mathcal{C}$ is a 'Yes'-instance of Set Cover if and only if $i_{\bar{\tau}}(K_n) \leq k$. Since the encoding length of the Set Cover instance $\mathcal{C}$ is $\Omega(\ln(k) + p|V|)$, the special choice of τ implies that the encoding length of (K_n, τ) is polynomially bounded in terms of the encoding length of $\mathcal{C}$.

Let $n' = 2p+1+|V|$, and let $V(K_{n'}) = \{S_1, \ldots, S_p\} \cup V \cup X$, where X is a set of order $p+1$ that is disjoint from $\{S_1, \ldots, S_p\} \cup V$. For $i \in [p]$, let $\tau'(S_i) = \{X\}$, for $u \in V$, let $\tau'(u) = \{\{S_i\} : i \in [p] \text{ and } u \in S_i\}$, and, for $x \in X$, let $\tau'(x) = \{V\}$. Again, the encoding length of $(K_{n'}, \tau')$ is polynomially bounded in terms of the encoding length of $\mathcal{C}$.

Suppose that $\mathcal{C}$ is a 'Yes'-instance of Set Cover, and let C be as above. Let $S = \{S_i : i \in C\}$. The definition of τ' implies that $I_{\bar{\tau}'}(S) = S \cup V$, $I^2_{\bar{\tau}'}(S) = S \cup V \cup X$, and $I^3_{\bar{\tau}'}(S) = V(K_{n'})$, which implies $h_{\bar{\tau}'}(K_{n'}) \leq k$.

Conversely, suppose that $h_{\bar{\tau}'}(K_{n'}) \leq k$. Let S be a set of at most k vertices of $K_{n'}$ with $H_{\bar{\tau}'}(S) = V(K_{n'})$ such that $|S \cap V|$ is as small as possible. If $u \in S \cap V$ and S_i is such that $u \in S_i$, then $u \in I_{\bar{\tau}'}((S \setminus \{u\}) \cup \{S_i\})$, which implies $H_{\bar{\tau}'}((S \setminus \{u\}) \cup \{S_i\}) = V(K_{n'})$. Since $|(S \setminus \{u\}) \cup \{S_i\}| \leq |S|$ and $|((S \setminus \{u\}) \cup \{S_i\}) \cap V| < |S \cap V|$, this yields a contradiction to the choice of S. Hence S contains no element of V. Since $k < p$, the set S does not contain all elements of X. If $V \not\subseteq I_{\bar{\tau}'}(S)$, then the definition of τ' implies $I_{\bar{\tau}'}(S) = H_{\bar{\tau}'}(S) \neq V(K_{n'})$, which is a contradiction. Hence, $V \subseteq I_{\bar{\tau}'}(S)$. We obtain $\bigcup_{S_i \in S} S_i = V$, which yields that $\mathcal{C}$ is a 'Yes'-instance of Set Cover. □

As observed by Chlebík and Chlebíková [7], Feige [20], Dinur and Steurer [15] showed the hardness of approximation of Set Cover for instance with $\ln(p + |V|) \approx \ln(|V|)$. Therefore, the construction in the proof of Theorem 1 actually implies the following.

Theorem 2 *For a given threshold function τ on a complete graph K_n, neither $i_{\bar{\tau}}(K_n)$ nor $h_{\bar{\tau}}(K_n)$ can be approximated in polynomial time within a factor of $(1-\epsilon)\ln(n)$ for any constant $\epsilon > 0$ unless P=NP.*

Replacing Set Cover with Exact Cover by 3-Sets (cf. the proof of Theorem 5 below) within the proof of Theorem 1 yields the following additional result.

Theorem 3 *Given a pair (k, τ), where k is a non-negative integer and τ is a threshold function on a complete graph K_n, it is NP-complete to decide whether $i_\tau(K_n) \leq k$.*

Note that in the construction used in the proof of Theorem 1 several edges of the complete graph K_n are in fact irrelevant because of the special choice of τ. Removing these edges easily implies that deciding $i_{\bar{\tau}}(G) \leq k$ and $i_\tau(G) \leq k$ are NP-complete for a given triple (G, k, τ), where G is a bipartite graph, k is a non-negative integer, and τ is a threshold function on G.

We present some more hardness results concerning special choices of the threshold function.

Theorem 4 *Given a graph G and a given integer k, it is NP-complete to decide whether $i_\tau(G) \leq k$, where τ is the threshold function on G with $\tau(u) = \binom{N_G(u)}{1}$ for every vertex u of G.*

Proof: Again, the considered decision problem is clearly in NP. In order to prove completeness, we describe a polynomial reduction from 1-in-3Sat. Therefore, let $\mathcal{F}$ be an instance of 1-in-3Sat with m clauses $C_1, \ldots, C_m$ over n Boolean variables $x_1, \ldots, x_n$. We specify a graph G and an integer k such that the order of G is polynomially bounded in terms of n and m, and $\mathcal{F}$ has a satisfying truth assignment that leads to exactly one true literal in each clause if and only if $i_\tau(G) \leq k$ for the special threshold function τ described in the statement.

Therefore,

- for every variable x_i, we create a copy G_i of $K_4 - e$ and denote the two vertices of degree 3 in G_i by x_i and $\bar{x}_i$,
- for every clause C_j, we create a vertex C_j, and

- for every literal $x \in \{x_1, \ldots, x_n\} \cup \{\bar{x}_1, \ldots, \bar{x}_n\}$ and every clause C_j such that x appears in C_j, we add the edge xC_j.

This completes the construction of G. Let $k = n$.

If $\mathcal{F}$ has a satisfying truth assignment that leads to exactly one true literal in each clause, then the set S of all vertices that correspond to true literals is a τ-interval set of order n. Conversely, if S is a τ-interval set of order at most n, then the special choice of τ and k implies that $S \cap V(G_i) \in \{\{x_i\}, \{\bar{x}_i\}\}$ for every $i \in [n]$, and $S \cap \{C_1, \ldots, C_m\} = \emptyset$. Therefore, since S is a τ-interval set, the elements in S indicate a satisfying truth assignment for $\mathcal{F}$ that leads to exactly one true literal in each clause, which completes the proof. $\Box$

Theorem 5 *Given a bipartite graph G and a given integer k, it is NP-complete to decide whether $i_\oplus(G) \leq k$.*

Proof: Again, the considered decision problem is clearly in NP. In order to prove completeness, we describe a polynomial reduction from Exact Cover by 3-Sets. Therefore, let $\mathcal{C}$ be an instance of Exact Cover by 3-Sets consisting of m 3-element subsets $C_1, \ldots, C_m$ of a ground set X of order $3n$. Clearly, we may assume that $\bigcup_{i=1}^m C_i = X$. We specify a bipartite graph G and an integer k such that the order of G is polynomially bounded in terms of n and m, and $\mathcal{C}$ is a 'Yes'-instance of Exact Cover by 3-Sets if and only if $i_\oplus(G) \leq k$. Let $V(G) = X \cup \{C_1, \ldots, C_m\} \cup \{r\} \cup \{s_1, \ldots, s_{n+2}\}$,

$$\begin{aligned} E(G) &= \{xC_i : x \in X, i \in [m], \text{ and } x \in C_i\} \cup \{rC_i : i \in [m]\} \cup \{rs_j : j \in [n+2]\}, \text{ and} \\ k &= n+1. \end{aligned}$$

If $\mathcal{C}$ is a 'Yes'-instance, and I is a set of exactly n indices in $[m]$ such that $\bigcup_{i \in I} C_i = X$, then $\{r\} \cup \{C_i : i \in I\}$ is an $\oplus$-interval set of G, which implies $i_\oplus(G) \leq n+1$. Conversely, let S be an $\oplus$-interval set of G of order at most $n+1$. If S does not contain r, then $s_j \in S$ for every $j \in [n+2]$, which implies the contradiction $|S| > n+1$. Hence, $r \in S$. Let $n_X = |S \cap X|$ and $n_C = |S \cap \{C_1, \ldots, C_m\}|$. Since each of the $3n - n_X$ vertices in $X \setminus S$ has at least one neighbor in $S \cap \{C_1, \ldots, C_m\}$, and every vertex C_i has exactly 3 neighbors in X, we obtain $3n - n_X \leq 3n_C$, which implies $\frac{1}{3}n_X + n_C \geq n$. Since $n_X + n_C \leq |S| - 1 \leq n$, we obtain $n_X = 0$. Therefore, if $I = \{i \in [m] : C_i \in S\}$, then $|I| = n_C \leq n$ and $\bigcup_{i \in I} C_i = X$, which implies that $\mathcal{C}$ is a 'Yes'-instance. $\Box$

2.2 Efficient algorithms for trees

In view of the strong hardness results in the previous subsection, efficient algorithms seem possible only for either quite restricted graph classes (and general threshold functions) or quite restricted threshold functions (and more general graph classes). The second of these two options comprises the many well known efficient algorithms for domination, multiple domination, vertex cover, independence and target set selection in special graph classes. Therefore, we focus here on the first option, that is, we do not want to restrict the threshold functions.

In [10, 11] generalizations of *target selection* problem (a special case of the problem studied here) have been studied on trees where size/cost set bound and time bound are considered. They provided linear algorithms for such cases. However these problems assume that any vertex v has an integer value $t(v)$ meaning that v needs $t(v)$ activate neighbors to become activate. Clearly relaxed τ-threshold process provides more general problems such as relaxed τ-interval (time bound equals 1) and relaxed τ-hull (time bound equals $n(G)$).

Relaxed and non-relaxed τ-hull number

We present some efficient algorithms for trees. Our first result concerns the relaxed τ-hull number.

Lemma 6 *Let G be a graph, and let τ be a threshold function on G. Let uv be an edge of G such that v has degree 1 in G.*

(i) If $\tau(v)=\emptyset$, then for the threshold function σ on $G-v$ defined as

$$\sigma(x)=\begin{cases}\{N\setminus\{v\}:N\in\tau(u)\} & \text{, if } x=u, \text{ and}\\ \tau(x) & \text{, otherwise,}\end{cases}$$

it holds that $h_{\bar{\tau}}(G)=h_{\bar{\sigma}}(G-v)+1$.

(ii) If $\emptyset\in\tau(v)$, and the threshold function σ on $G-v$ is as in (i), then $h_{\bar{\tau}}(G)=h_{\bar{\sigma}}(G-v)$.

(iii) If $\tau(v)=\{\{u\}\}$, and the threshold function σ on $G-v$ is such that

$$\sigma(x)=\begin{cases}\{N:N\in\tau(u) \text{ and } v\notin N\} & \text{, if } x=u, \text{ and}\\ \tau(x) & \text{, otherwise,}\end{cases}$$

then $h_{\bar{\tau}}(G)=h_{\bar{\sigma}}(G-v)$.

Proof: (i) Since $\bar{\tau}(v)=\emptyset$, every $\bar{\tau}$-hull set of G contains v. The definition of σ easily implies that S is a $\bar{\tau}$-hull set of G if and only if $S\setminus\{v\}$ is a $\bar{\sigma}$-hull set of $G-v$, which implies (i).

(ii) Since $\emptyset\in\bar{\tau}(v)$, no $\bar{\tau}$-hull set of G that is minimal with respect to inclusion contains v. For a set $S\subseteq V(G)\setminus\{v\}$, the definition of σ easily implies that S is a $\bar{\tau}$-hull set of G if and only if S is a $\bar{\sigma}$-hull set of $G-v$, which implies (ii).

(iii) If S is a $\bar{\tau}$-hull set of G that is minimal with respect to inclusion and $v\in S$, then $u\notin S$, and $(S\setminus\{v\})\cup\{u\}$ is a $\bar{\tau}$-hull set of G. Hence, G has a $\bar{\tau}$-hull set of minimum order that does not contain v. For a set $S\subseteq V(G)\setminus\{v\}$, the definition of σ easily implies that S is a $\bar{\tau}$-hull set of G if and only if S is a $\bar{\sigma}$-hull set of $G-v$, which implies (iii). $\square$

Iteratively applying Lemma 6 immediately implies the following.

Theorem 7 *For a given pair (T,τ), where T is a tree and τ is a threshold function on T, the relaxed τ-hull number of T can be determined in linear time.*

Note that the linear running time refers to the encoding length as specified in (1).

The non-relaxed τ-hull number seems to be much harder even for trees. More specifically, given a tree T rooted by some vertex u and a threshold function τ on T, it seems difficult to verify even if u does not belong to any τ-hull set S of T. This occurs because u needs an exact subset $N\in\tau(u)$ of its neighbors at some time step $t\geq 1$ during the process, such that $N\subseteq H(S)$ at t and all of the vertices in $N_G(u)\setminus N$ are not in $H(S)$ at t. Hence, it is required a mutual synchronization of the τ-hull sets of all subtrees of u. However, a simple dynamic programming approach works for paths.

Proposition 8 *For a given pair (P_n,τ), where τ is a threshold function on the path P_n of order n, the τ-hull number of P_n can be determined in polynomial time.*

Proof: For every two vertices u and v of P_n, it is possible to determine efficiently whether $H_\tau(\{u,v\})$ contains all vertices that lie between u and v on P_n. Similarly, for every vertex u of P_n, it is possible to determine efficiently whether $H_\tau(\{u\})$ contains all vertices that lie between u and some specific endvertex of P_n. In view of these observations, a simple dynamic programming approach allows to determine the τ-hull number of P_n efficiently. $\square$

Relaxed and non-relaxed τ-interval number

Remember that we cannot able to answer whether a given vertex does not belong to a τ-hull set of a tree T due to the requirement of a synchronization of the τ-hull sets of the subtrees of a vertex. The next two results concern the recognition of vertices that are not in every τ-interval set for trees.

Lemma 9 *Let T be a tree, and let τ be a threshold function on T. Let u be a vertex. For $v \in N_T(u)$, let T_v be the component of $T - u$ that contains v.*

There is a τ-interval set S of T that does not contain u if and only if there is some set N in $\tau(u)$ such that for every $v \in N_G(u) \setminus N$, there is a τ_v-interval set S_v of T_v that does not contain v, where

$$\tau_v(x) = \begin{cases} \{X : X \in \tau(v) \text{ and } u \notin X\} & \text{, if } x = v \text{, and} \\ \tau(x) & \text{, } x \in V(T_v) \setminus \{v\}. \end{cases}$$

Proof: Suppose that S is a τ-interval set of T that does not contain u. Clearly, $N = S \cap N_G(u) \in \tau(u)$, and if $v \in N_G(u) \setminus N$, then $S \cap V(T_v)$ is a τ_v-interval set of T_v that does not contain v. This proves the necessity.

For the sufficiency, let $N \in \tau(u)$, and S_v for $v \in N_G(u) \setminus N$ be as in the statement. By the definition of τ_v, the set $S = \bigcup_{v \in N} V(T_v) \cup \bigcup_{v \in N_G(u) \setminus N} S_v$ is a τ-interval set of T that does not contain u. $\square$

Proposition 10 *For a given triple (T, τ, u), where T is a tree, τ is a threshold function on T, and u is a vertex of T, one can determine in linear time whether T has a τ-interval set that does not contain u.*

Proof: We root T in u. For every vertex v of T that is distinct from u, let T_v denote the subtree of T that contains v as well as all descendants of v, and that is rooted in v. Furthermore, if v^- is the parent of v, then let τ_v be the threshold function on T_v defined by

$$\tau_v(x) = \begin{cases} \{X : X \in \tau(v) \text{ and } v^- \notin X\} & \text{, if } x = v \text{, and} \\ \tau(x) & \text{, } x \in V(T_v) \setminus \{v\}. \end{cases}$$

Iteratively applying Lemma 9 and processing the vertices v of T in an order of non-increasing depth, one can determine in linear time, whether T_v has a τ_v-interval set that does not contain v. Note that if exists $N \in \tau(u)$ such that for every $v \in N_G(u) \setminus N$, T_v has a τ-interval set that does not contain v, then T has a τ-interval set S that does not contain u, because, without loss of generality, $\bigcup_{w \in N} V(T_w) \subseteq S$. $\square$

Our next goal is an efficient algorithm for the τ-interval number of a tree.

Lemma 11 *Let T be a rooted tree, and let τ be a threshold function on T. For every vertex u of T that is not a leaf, every child v of u, and every subset F of $\{u, v\}$, let*

$$i(u, v, F) = \min \Big\{ |S \setminus \{u\}| : S \text{ is a } \tau_{uv}\text{-interval set of } T_{uv} \text{ with } S \cap \{u, v\} = F \Big\},$$

where T_{uv} is the subtree of T that contains u, v, and all descendants of v, τ_{uv} is the threshold function on T_{uv} such that

$$\tau_{uv}(x) = \begin{cases} \{F \cap \{v\}\} & \text{, if } x = u \text{, and} \\ \tau(x) & \text{, } x \in V(T_{uv}) \setminus \{u\}, \end{cases}$$

and $\min \emptyset = \infty$.

(i) If v is a leaf, then

$$i(u, v, F) = \begin{cases} 1 & \text{, if } v \in F, \\ 0 & \text{, if } v \notin F \text{ and } F \cap \{u\} \in \tau(v) \text{, and} \\ \infty & \text{, otherwise.} \end{cases}$$

(ii) If v is not a leaf, W is the set of children of v, and $v \notin F$, then

$$i(u, v, F) = \min \left\{ \sum_{w \in W \cap N} i(v, w, \{w\}) + \sum_{w \in W \setminus N} i(v, w, \emptyset) : N \in \tau(v) \text{ with } N \cap \{u\} = F \cap \{u\} \right\}.$$

(iii) If v is not a leaf, W is the set of children of v, and $v \in F$, then

$$i(u,v,F) = 1 + \sum_{w \in W} \min\Big\{i(v,w,\{v\}), i(v,w,\{v,w\})\Big\}.$$

Proof: (i) Note that $V(T_{uv}) = \{u,v\}$. The set F is a τ_{uv}-interval set of T_{uv} if and only if either $v \in F$ or $v \not\in F$ but $F \cap \{u\} \in \tau(v)$. By the definition of $i(u,v,F)$, this immediately yields the stated values.

(ii) Let S be a τ_{uv}-interval set of T_{uv} with $S \cap \{u,v\} = F$ and $i(u,v,F) = |S \setminus \{u\}|$. Since $v \not\in F$, we obtain that $v \not\in S$. Hence, the set $N = S \cap N_{T_{uv}}(v)$ belongs to $\tau(v)$ and satisfies $N \cap \{u\} = F \cap \{u\}$. For every child w of v, let $S_w = S \cap V(T_{vw})$. If $w \in W \cap N$, then S_w is a τ_{vw}-interval set of T_{vw} with $S_w \cap \{v,w\} = \{w\}$, and, if $w \in W \setminus N$, then S_w is a τ_{vw}-interval set of T_{vw} with $S_w \cap \{v,w\} = \emptyset$. This implies

$$i(u,v,F) \;=\; |S \setminus \{u\}| = \sum_{w \in W \cap N} |S_w| + \sum_{w \in W \setminus N} |S_w| \geq \sum_{w \in W \cap N} i(v,w,\{w\}) + \sum_{w \in W \setminus N} i(v,w,\emptyset),$$

which implies that $i(u,v,F)$ is at least the minimum stated in (ii).

Conversely, let $N \in \tau(v)$ with $N \cap \{u\} = F \cap \{u\}$ be such that

$$\sum_{w \in W \cap N} i(v,w,\{w\}) + \sum_{w \in W \setminus N} i(v,w,\emptyset)$$

is minimum. For $w \in W \cap N$, let S_w be a τ_{vw}-interval set of T_{vw} with $S_w \cap \{v,w\} = \{w\}$ and $i(v,w,\{w\}) = |S_w \setminus \{v\}|$, and, for $w \in W \setminus N$, let S_w be a τ_{vw}-interval set of T_{vw} with $S_w \cap \{v,w\} = \emptyset$ and $i(v,w,\emptyset) = |S_w \setminus \{v\}|$. By the definition of τ_{uv} and the choice of N, the set $S = F \cup \bigcup_{w \in W} S_w$ is a τ_{uv}-interval set of T_{uv} with $S \cap \{u,v\} = F$. We obtain

$$i(u,v,F) \;\leq\; |S \setminus \{u\}| = \sum_{w \in W \cap N} |S_w| + \sum_{w \in W \setminus N} |S_w| = \sum_{w \in W \cap N} i(v,w,\{w\}) + \sum_{w \in W \setminus N} i(v,w,\emptyset),$$

which implies that $i(u,v,F)$ is at most the minimum stated in (ii).

(iii) Let S be a τ_{uv}-interval set of T_{uv} with $S \cap \{u,v\} = F$ and $i(u,v,F) = |S \setminus \{u\}|$. For every child w of v, let $S_w = S \cap V(T_{vw})$. Since $v \in F \subseteq S$, the set S_w is a τ_{vw}-interval set of T_{vw} with $v \in S_w$. We obtain

$$i(u,v,F) \;=\; |S \setminus \{u\}| = |\{v\}| + \sum_{w \in W} |S_w \setminus \{v\}| \geq 1 + \sum_{w \in W} \min\Big\{i(v,w,\{v\}), i(v,w,\{v,w\})\Big\},$$

which implies that $i(u,v,F)$ is at least the minimum stated in (iii).

Conversely, for $w \in W$, let S_w be a τ_{vw}-interval set of T_{vw} with $v \in S_w$ and

$$\min\Big\{i(v,w,\{v\}), i(v,w,\{v,w\})\Big\} = |S_w \setminus \{v\}|.$$

By the definition of τ_{uv}, the set $S = F \cup \bigcup_{w \in W} S_w$ is a τ_{uv}-interval set of T_{uv} with $S \cap \{u,v\} = F$. We obtain

$$i(u,v,F) \;\leq\; |S \setminus \{u\}| = |\{v\}| + \sum_{w \in W} |S_w \setminus \{v\}| = 1 + \sum_{w \in W} \min\Big\{i(v,w,\{v\}), i(v,w,\{v,w\})\Big\},$$

which implies that $i(u,v,F)$ is at most the minimum stated in (iii). □

Theorem 12 *For a given pair (T,τ), where T is a tree of order n and τ is a threshold function on T, the τ-interval number of T can be determined in linear time.*

Proof: We root T in an endvertex r. Let s be the neighbor of r in T. By Lemma 11, the vectors $\Big(i(u,v,\emptyset), i(u,v,\{u\}), i(u,v,\{v\}), i(u,v,\{u,v\})\Big)$ can be determined in linear time processing the edges uv of T in an order of non-increasing depth of u. By definition,

$$i_\tau(T) = \min\Big\{ i(r,s,F) + |F \cap \{r\}| : F \subseteq \{r,s\} \text{ with either } r \in F \text{ or } r \notin F \text{ and } F \cap \{s\} \in \tau(r). \Big\}.$$

□

By definition, the relaxed τ-interval number of a tree T equals the $\bar{\tau}$-interval number of T, and we can apply the algorithm described in the proof of Theorem 12. Unfortunately, this only leads to a running time that is linear in the encoding length of $(T, \bar{\tau})$, which may be much bigger than the encoding length of (T, τ). Therefore, for the next result, we need to argue how to obtain a running time that is linear in the encoding length of (T, τ).

Corollary 13 *For a given pair (T, τ), where T is a tree of order n and τ is a threshold function on T, the relaxed τ-interval number of T can be determined in linear time.*

Proof: In view of Lemma 11, it suffices to argue, how to evaluate the expression

$$\min\left\{ \sum_{w \in W \cap \bar{N}} i(v,w,\{w\}) + \sum_{w \in W \setminus \bar{N}} i(v,w,\emptyset) : \bar{N} \in \bar{\tau}(v) \text{ with } \bar{N} \cap \{u\} = F \cap \{u\} \right\}$$

in $O(d_G(v)|\tau(v)|)$ time, where v is a child of u, and $v \notin F \subseteq \{u,v\}$. By the definition of $\bar{\tau}$, we obtain that $\bar{N} \in \bar{\tau}(v)$ if and only if there is a set $N \in \tau(v)$ with $N \subseteq \bar{N}$, which easily implies that the above expression equals

$$\min\left\{ \sum_{w \in W \cap N} i(v,w,\{w\}) + \sum_{w \in W \setminus N} \min\{i(v,w,\emptyset), i(v,w,\{w\})\} : N \in \tau(v) \text{ with } N \cap \{u\} \subseteq F \cap \{u\} \right\},$$

which can clearly be determined in $O(d_G(v)|\tau(v)|)$ time. □

The problem of computing a smallest $\oplus$-interval set is close related to parity domination problem [21], which can be solved in linear time for graphs with bounded treewidth [21]. However, parity domination algorithms cannot be applied to obtain the $\oplus$-interval number of a graph, because the parity domination set property depends on a given partition of the vertices and a given function $\pi : V \to \{0,1\}$. For more information about parity domination see [14, 21, 22].

Hence, in order to determine the $\oplus$-interval number in linear time, the special choice $\tau = \oplus$ requires some arguments concerning the running time.

Theorem 14 *For a given tree T, the $\oplus$-interval number of T can be determined in linear time.*

Proof: Again, we can root T in an endvertex r and apply the algorithm described in the proof of Theorem 12. Now, we only need to argue how to improve its running time to be linear in the encoding length of T. Consider w^- the parent of a vertex w of T. In view of Lemma 11, it suffices to explain how to minimize an expression of the form

$$\sum_{w \in N} i_1(w) + \sum_{w \in W \setminus N} i_2(w)$$

over all subsets N of a set W of a fixed parity p modulo 2 in $O(|W|)$ time, where

$$i_1(w) = \left| \sum_{s \in N(w) \setminus \{w^-\}} \min\Big\{ i(w,s,F) : F \subseteq \{r,s\} \text{ and } r \in F \Big\} \right| + 1,$$

and

$$i_2(w) = \left| \sum_{s\in N(w)\setminus\{w^-\}} \min\left\{ i(w,s,F) : F\subseteq \{r,s\} \text{ and } r\not\in F \right\} \right|.$$

If $i_1(w) = i_2(w)$ for some $w \in W$, then the minimum equals $\sum_{w\in W} \min\{i_1(w), i_2(w)\}$. Hence, we may assume that $i_1(w) \neq i_2(w)$ for every $w \in W$. Let $N = \{w \in W : i_1(w) < i_2(w)\}$. If $|N| \mod 2 = p$, then the minimum equals again $\sum_{w\in W} \min\{i_1(w), i_2(w)\}$. If $|N| \mod 2 \neq p$, and $w^* \in W$ is such that

$$|i_1(w^*) - i_2(w^*)| = \min\{|i_1(w) - i_2(w)| : w \in W\},$$

then the minimum equals $|i_1(w^*)-i_2(w^*)|+\sum_{w\in W} \min\{i_1(w), i_2(w)\}$. These observations easily imply the linear running time. □

2.3 A probabilistic upper bound

It seems intuitively plausible that large average cardinalities of the sets $\tau(u)$ should lead to small values of the corresponding relaxed or non-relaxed interval and hull numbers. The following result supports this intuition at least for the relaxed τ-interval number. We then describe an example showing that this intuition is actually misleading when it comes to the (non-relaxed) τ-interval number.

Theorem 15 *If G is a graph of order n, τ is a threshold function on G, and $0 \leq p \leq 1$, then*

$$i_{\bar{\tau}}(G) \leq n - \sum_{u\in V(G)} \sum_{i=0}^{d_G(u)} \left| \bar{\tau}(u) \cap \binom{N_G(u)}{i} \right| p^i(1-p)^{d_G(u)-i+1}.$$

Proof: Let X be a random subset of $V(G)$ that contains each vertex of G independently at random with probability p. Let $Y = \{u \in V(G) : u \not\in X \text{ and } N_G(u) \cap X \not\in \bar{\tau}(u)\}$. By the choice of X, for every vertex u of G, the probability that $N_G(u) \cap X$ belongs to $\bar{\tau}(u)$ equals

$$p(u) := \sum_{i=0}^{d_G(u)} \left| \bar{\tau}(u) \cap \binom{N_G(u)}{i} \right| p^i(1-p)^{d_G(u)-i}.$$

This implies that $\mathbf{P}[u \in Y] = (1-p)(1-p(u))$. Since $X \cup Y$ is a $\bar{\tau}$-interval set, we obtain

$$\begin{aligned} i_{\bar{\tau}}(G) &\leq \mathbf{E}[|X| + |Y|] \\ &= \mathbf{E}[|X|] + \sum_{u\in V(G)} \mathbf{P}[u \in Y] \\ &= pn + \sum_{u\in V(G)} (1-p)(1-p(u)) \\ &= n - \sum_{u\in V(G)} (1-p)p(u) \end{aligned}$$

which completes the proof. □

For $p = \frac{1}{2}$, the term in Theorem 15 simplifies as follows.

Corollary 16 *If G is a graph of order n, and τ is a threshold function on G, then*

$$i_{\bar{\tau}}(G) \leq n - \frac{1}{2} \sum_{u\in V(G)} \frac{|\bar{\tau}(u)|}{2^{d_G(u)}}.$$

Using the method of conditional expectation, it is possible to devise an efficient deterministic algorithm based on the proof of Theorem 15 that produces a $\bar{\tau}$-interval set of the guaranteed cardinality for a given pair (G, τ).

The following example suggests that there is no reasonable version of Corollary 16 for the non-relaxed τ-interval number. Let k and n be positive integers with $n > k$. Let (G,τ) be such that G is a complete graph with vertex set $V(G) = \{x_1,\ldots,x_k\} \cup \{y_1,\ldots,y_{n-k}\}$, and τ is a threshold function on G defined as follows:

$$u \mapsto \begin{cases} \{\{y_1,\ldots,y_{n-k}\}\} & \text{, if } u \in \{x_1,\ldots,x_k\}\text{, and} \\ \left\{N \subseteq N_G(u) : \{x_1,\ldots,x_k\} \not\subseteq N\right\} & \text{, if } u \in \{y_1,\ldots,y_{n-k}\}. \end{cases}$$

Let I be a τ-interval set of G. If $\{x_1,\ldots,x_k\} \not\subseteq I$, then the value of $\tau(x_i)$ for some $x_i \notin I$ implies $\{y_1,\ldots,y_{n-k}\} \subseteq I$. If $\{x_1,\ldots,x_k\} \subseteq I$, then the value of $\tau(y_j)$ for every y_j implies $\{y_1,\ldots,y_{n-k}\} \subseteq I$. This easily implies $i_\tau(G) = n-k$. Note that

$$\sum_{u\in V(G)} \frac{|\tau(u)|}{2^{d_G(u)}} = \sum_{i=1}^{k} \frac{1}{2^{n-1}} + \sum_{j=1}^{n-k} \frac{2^{n-k-1}\left(2^k-1\right)}{2^{n-1}} > \left(1-2^{-k}\right)(n-k).$$

This implies that for every $\epsilon > 0$, there is a pair (G,τ), where G has order n, such that

$$\sum_{u\in V(G)} \frac{|\tau(u)|}{2^{d_G(u)}} \geq (1-\epsilon)n \quad \text{but} \quad i_\tau(G) \geq (1-\epsilon)n,$$

that is, even though on average each of the sets $\tau(u)$ contains at least a $(1-\epsilon)$-fraction of all $2^{d_G(u)}$ possible subsets of $N_G(u)$, the τ-interval number is close to the trivial upper bound n.

3 Conclusion

We conclude with several open questions.

What is the complexity of the non-relaxed τ-hull number for trees? Are there non-trivial bounds on the relaxed or non-relaxed τ-hull number?

It seems interesting to study a 'total' version of τ-interval sets, where S is a *total τ-interval set* of a graph G with threshold function τ if $N_G(u) \cap S \in \tau(u)$ for every vertex u of G, and not only for every vertex u in $V(G) \setminus S$. It is easy to see that deciding the existence of a total $\oplus$-interval set for a given graph is equivalent to a suitable system of n linear equalities over $\mathbb{F}_2$.

Let V be a set of order n. For which vectors $(t_0, t_1, \ldots, t_n)$ does there exist a set system $\mathcal{T} \subseteq 2^V$ with $\forall X \in \mathcal{T} : \forall Y \subseteq V : X \subseteq Y \Rightarrow Y \in \mathcal{T}$ such that $t_i = \left|\mathcal{T} \cap \binom{V}{i}\right|$ for every i?

References

[1] E. Ackerman, O. Ben-Zwi, G. Wolfovitz, Combinatorial model and bounds for target set selection, Theoretical Computer Science 411 (2010) 4017-4022.

[2] M. Aizenman, J.L. Lebowitz, Metastability effects in bootstrap percolation, Journal of Physics A 21 (1988) 3801-3813.

[3] P. Balister, B. Bollobás, J.R. Johnson, M. Walters, Random majority percolation, Random Structures and Algorithms 36 (2010) 315-340.

[4] J.-C. Bermond, J. Bond, D. Peleg, S. Perennes, The power of small coalitions in graphs, Discrete Applied Mathematics 127 (2003) 399-414.

[5] N. Chen, On the approximability of influence in social networks, SIAM Journal on Discrete Mathematics 23 (2009) 1400-1415.

[6] J.R. Calder, Some elementary properties of interval convexities, Journal of the London Mathematical Society 3 (1971) 422-428.

[7] M. Chlebík, J. Chlebíková, Approximation hardness of dominating set problems in bounded degree graphs, Information and Computation 206 (2008) 1264-1275.

[8] C.C. Centeno, M.C. Dourado, L.D. Penso, D. Rautenbach, J.L. Szwarcfiter, Irreversible conversion of graphs, Theoretical Computer Science 412 (2011) 3693-3700.

[9] E.J. Cockayne, B. Gamble, B. Shepherd, An upper bound for the k-domination number of a graph, Journal of Graph Theory 9 (1985) 533-534.

[10] F. Cicalese, G. Cordasco, L. Gargano, M. Milanič, U. Vaccaro, Latency-bounded target set selection in social networks, Theoretical Computer Science 535 (2014) 1-15.

[11] F. Cicalese, G. Cordasco, L. Gargano, M. Milanič, J. Peters, U. Vaccaro, Spread of influence in weighted networks under time and budget constraints, Theoretical Computer Science 586 (2015) 40-58.

[12] F. Cicalese, Ferdinando, M. Milanič, U. Vaccaro. On the approximability and exact algorithms for vector domination and related problems in graphs. Discrete Applied Mathematics 161.6 (2013) 750-767.

[13] C.C. Centeno, D. Rautenbach, Remarks on dynamic monopolies with given average thresholds, Discussiones Mathematicae Graph Theory 35 (2015) 133-140.

[14] R. Dawes, Minimum odd neighbourhood covers for trees, Lect Notes Comput Sci 507 (1989) 161-169.

[15] I. Dinur, D. Steurer, Analytical approach to parallel repetition. In: Proceedings of the 46th Annual ACM Symposium on Theory of Computing (2014) 624-633.

[16] P.A. Dreyer Jr., F.S. Roberts, Irreversible k-threshold processes: Graph-theoretical threshold models of the spread of disease and of opinion, Discrete Applied Mathematics 157 (2009) 1615-1627.

[17] P. Domingos, M. Richardson, Mining the network value of customers, in Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2001, 57-66.

[18] M.C. Dourado, D. Rautenbach, V.F. dos Santos, P.M. Schäfer, J.L. Szwarcfiter, On the Carathéodory number of interval and graph convexities, Theoretical Computer Science 520 (2013) 127-135.

[19] O. Favaron, A. Hansberg, L. Volkmann, On k-domination and minimum degree in graphs, Journal of Graph Theory 57 (2008) 33-40.

[20] U. Feige, A threshold of $\ln n$ for approximating set cover, Journal of the ACM 45 (1998) 634-652.

[21] E. Gassner, J. Hatzl, A parity domination problem in graphs with bounded treewidth and distance-hereditary graphs, Computing 82.2-3 (2008) 171-187.

[22] T.W. Haynes, S.T. Hedetniemi, P.J. Slater, Fundamentals of Domination in Graphs, Marcel Dekker, Inc., New York, 1998.

[23] D. Pradhan, Algorithmic aspects of k-tuple total domination in graphs, Information Processing Letters 112 (2012) 816-822. SIAM Journal on Computing 28 (1999) 526-541.

[24] D. Rautenbach, V. dos Santos, P.M. Schäfer, Irreversible conversion processes with deadlines, Journal of Discrete Algorithms 26 (2014) 69-76.

[25] D. Rautenbach, L. Volkmann, New bounds on the k-domination number and the k-tuple domination number, Applied Mathematics Letters 20 (2007) 98-102.

[26] M. Zaker, On dynamic monopolies of graphs with general thresholds, Discrete Mathematics 312 (2012) 1136-1143.

[27] V. Zverovich, On general frameworks and threshold functions for multiple domination, Discrete Mathematics 338 (2015) 2095-2104.

Appendix E:

Decycling with a Matching (Resume)

This appendix contains the resume "*Decycling with a Matching*" presented in *II Workshop Franco-brasileiro de Grafos e Otimização Combinatória*, 2016.

Decycling with a matching

Carlos V.G.C. Lima
and Jayme L. Szwarcfiter
PESC–COPPE,
UFRJ, Rio de Janeiro, Brazil.
Email: {gclima,jayme}@cos.ufrj.br

Dieter Rautenbach
Institute of Optimization and Operations Research,
Ulm University, Ulm, Germany.
Email: dieter.rautenbach@uni-ulm.de

Uéverton S. Souza
Institute of Computing,
UFF, Niterói, Brazil.
Email: ueverton@ic.uff.br

I. Introduction

Given a finite, simple, and undirected graph of order n and size m, destroying all cycles by removing vertices or edges is a classical theme. The minimum number of edges whose removal destroys all cycles is exactly $m - n + 1$, and standard minimum spanning tree algorithms allow to solve even weighted optimization versions. Contrary to this, the minimum number of vertices whose removal yields a tree is a difficult parameter [1]–[3], [5], [6].

We study the apparently simple case when the removed edges are required to form a matching. Quite surprisingly, we show that the corresponding decision problem, that is, the problem to decide whether a given graph is the union of a tree and a matching, is already hard. Furthermore, we present efficient algorithms for a number of well-known graph classes.

For a set E of edges of a graph G, let $G - E$ be the graph with vertex set $V(G)$ and edge set $E(G) \setminus E$. If $G - E$ is a forest, then E is *decycling*. Let $\mathcal{FM}$ be the set of all graphs that have a decycling matching.

II. Results

The following lemma collects some basic observations concerning graphs that have a decycling matching.

***Lemma** 1:* Let G be a graph.

(i) If $G \in \mathcal{FM}$ is connected, then G has a matching M for which $G - M$ is a tree.
(ii) If $G \in \mathcal{FM}$, then $m(H) \leq \left\lfloor \frac{3n(H)}{2} \right\rfloor - 1$ for every subgraph H of G.
(iii) If G is subcubic and connected, then $G \in \mathcal{FM}$ if and only if G has a spanning tree T such that all endvertices of T are of degree at most 2 in G.

Lemma 1(iii) is the key observation for the following result.

***Theorem** 2:* For a given 2-connected planar subcubic graph G, it is NP-complete to decide whether $G \in \mathcal{FM}$.

Sketch of Proof: The considered decision problem is clearly in NP. The 3-connected planar cubic graphs G constructed in [4] contain several edges that necessarily belong to every Hamiltonian cycle of G; regardless of whether such a cycle exists or not. Therefore, removing such an edge, their construction implies the NP-completeness of the following decision problem: *Given a 2-connected planar subcubic graph G with exactly two vertices u and v of degree 2, does G have a Hamiltonian path whose endvertices are u and v?*

Let G be a 2-connected planar subcubic graph with exactly two vertices u and v of degree 2. In order to complete the proof, it suffices to show that G has a Hamiltonian path whose endvertices are u and v if and only if $G \in \mathcal{FM}$. ■

We also consider a more general decision problem.

Allowed Decycling Matching

Instance:	A graph G and a set F of edges of G.
Task:	Decide whether G has a decycling matching M that does not intersect F, and determine such a matching if it exists.

For this new version, we summarize our positive results as follows.

***Theorem** 3:* Allowed Decycling Matching can be solved in polynomial time for $\{K_{1,3}, K_{1,3} + e\}$-free graphs.

***Theorem** 4:* Allowed Decycling Matching can be solved in polynomial time for P_5-free graphs.

***Theorem** 5:* Allowed Decycling Matching can be solved in polynomial time for chordal graphs.

***Theorem** 6:* Allowed Decycling Matching can be solved in polynomial time for C_4-free distance hereditary graphs.

III. Conclusion

We study a special case of the more general problem of destroying all cycles by removing edges under the restriction that the graph formed by the removed edges has bounded maximum degree. This problem can certainly be considered more generally. Furthermore, one can consider variants, such as, for instance, deciding whether a given graph is the union of a bipartite graph and a matching, that is, whether it has a matching whose removal destroys all odd cycles.

References

[1] S. Bau, L.W. Beineke, The decycling number of graphs, Australasian Journal of Combinatorics 25 (2002) 285-298.
[2] L.W. Beineke, R.C. Vandell, Decycling graphs, Journal of Graph Theory 25 (1997) 59-77.
[3] P. Erdős, M. Saks, V.T. Sós, Maximum induced trees in graphs, Journal Combinatorial Theory, Series B 41 (1986) 61-79.
[4] M.R. Garey, D.S. Johnson, R.E. Tarjan, The planar Hamiltonian circuit problem is NP-complete, SIAM Journal on Computing 5 (1976) 704-714.
[5] M. Gentner, D. Rautenbach, Feedback vertex sets in cubic multigraphs, Discrete Mathematics 338 (2015) 2179-2185.
[6] D. Rautenbach, Dominating and large induced trees in regular graphs, Discrete Mathematics 307 (2007) 3177-3186.

Appendix F:

Decycling with a Matching (Article)

This appendix contains the article “*Decycling with a Matching*” submitted on January 2015 to *Information Processing Letters* journal.

Decycling with a Matching

Carlos V.G.C. Lima[1]
Dieter Rautenbach[2]
Uéverton S. Souza[3]
Jayme L. Szwarcfiter[1]

[1] PESC, COPPE, Federal University of Rio de Janeiro, Rio de Janeiro, Brazil
{gclima,jayme}@cos.ufrj.br

[2] Institute of Optimization and Operations Research, Ulm University, Ulm, Germany
dieter.rautenbach@uni-ulm.de

[3] Institute of Computing, Fluminense Federal University, Niterói, Brazil
ueverton@ic.uff.br

Abstract

As a natural variant of the many decycling notions studied in graphs, we consider the problem to decide whether a given graph G has a matching M such that $G - M$ is a forest. We establish NP-completeness of this problem for 2-connected planar subcubic graphs, and describe polynomial time algorithms that also determine such a matching if it exists for graphs that are claw- and paw-free, P_5-free graphs, chordal graphs, and C_4-free distance hereditary graphs.

Keywords: decycling; matching; feedback vertex set

1 Introduction

We consider finite, simple, and undirected graphs, and use standard notation and terminology.

Destroying all cycles of a given graph by removing vertices or edges is a classical theme. Clearly, the minimum number of edges of a connected graph of order n and size m whose removal destroys all cycles is exactly $m - n + 1$, and standard minimum spanning tree algorithms allow to solve even weighted optimization versions. Contrary to this, the minimum number of vertices whose removal destroys all cycles (or produces a tree) is a difficult parameter [2–4, 6, 8].

In the present paper we study a special case of the problem of destroying all cycles by removing only edges under the natural restriction that the graph formed by the removed edges has bounded maximum degree. In fact, we consider the apparently simple case when the removed edges are required to form a matching. Quite surprisingly, we show that the corresponding decision problem, that is, the problem to decide whether a given graph is the union of a tree and a matching, is already hard. Furthermore, we present efficient algorithms for a number of well-known graph classes.

For a set E of edges of a graph G, let $G - E$ be the graph with vertex set $V(G)$ and edge set $E(G) \setminus E$. If $G - E$ is a forest, then E is *decycling*. Let $\mathcal{FM}$ be the set of all graphs that have a decycling matching.

2 Results

The following lemma collects some basic observations concerning graphs that have a decycling matching.

Lemma 1 *Let G be a graph.*

(i) If $G \in \mathcal{FM}$ is connected, then G has a matching M for which $G - M$ is a tree.

(ii) If $G \in \mathcal{FM}$, then $m(H) \leq \left\lfloor \frac{3n(H)}{2} \right\rfloor - 1$ for every subgraph H of G.

(iii) If G is subcubic and connected, then $G \in \mathcal{FM}$ if and only if G has a spanning tree T such that all endvertices of T are of degree at most 2 in G.

Proof: (i) Let $G \in \mathcal{FM}$ be connected. Let M be a matching of G such that $G - M$ is a forest F with as few components as possible. Suppose, for a contradiction, that F is not connected. Since G is connected, M contains an edge e between different components of F. Now, $N = M \setminus \{e\}$ is a matching of G such that $G - N$ is a forest with less components than F, which implies a contradiction. Hence, F is a tree.

(ii) Let $G \in \mathcal{FM}$. Since $\mathcal{FM}$ is closed under taking subgraphs, it suffices to show $m(G) \leq \left\lfloor \frac{3n(G)}{2} \right\rfloor - 1$. Let M be a decycling matching of G. Clearly, $m(G) \leq m(G - M) + |M| \leq (n(G) - 1) + \left\lfloor \frac{n(G)}{2} \right\rfloor = \left\lfloor \frac{3n(G)}{2} \right\rfloor - 1$.

(iii) Let G be a connected subcubic graph. Clearly, we may assume that $n(G) \geq 3$.

First, suppose that $G \in \mathcal{FM}$. By (i), G has a matching M such that $G - M$ is a spanning tree T. If u is an endvertex of T, then $d_G(u) \leq d_T(u) + 1 \leq 2$, which implies that all endvertices of T have degree at most 2 in G.

Next, suppose that T is a spanning tree of G such that all endvertices of T are of degree at most 2 in G. Let $M = E(G) \setminus E(T)$. Clearly, M is decycling, and it remains to show that M is a matching. Suppose that M contains two edges incident with the same vertex u of G. This implies $d_T(u) \leq d_G(u) - 2 \leq 3 - 2 = 1$, that is, u is an endvertex of T. By the choice of T, we obtain $d_T(u) \leq d_G(u) - 2 \leq 2 - 2 = 0$, which is a contradiction. $\square$

Lemma 1(iii) is the key observation for the following hardness result.

Theorem 2 *For a given 2-connected planar subcubic graph G, it is NP-complete to decide whether $G \in \mathcal{FM}$.*

Proof: The considered decision problem is clearly in NP. In order to show NP-completeness, we use [5] that deciding the existence of a Hamiltonian cycle for a given 3-connected planar cubic graph is NP-complete. In fact, the 3-connected planar cubic graphs G constructed by Garey et al. in [5] contain several edges that necessarily belong to every Hamiltonian cycle of G; regardless of whether such a cycle exists or not. Therefore, removing such an edge, their construction implies the NP-completeness of the following decision problem: *Given a 2-connected planar subcubic graph G with exactly two vertices u and v of degree 2, does G have a Hamiltonian path whose endvertices are u and v?*

Let G be a 2-connected planar subcubic graph with exactly two vertices u and v of degree 2. In order to complete the proof, it suffices to show that G has a Hamiltonian path whose endvertices are u and v if and only if $G \in \mathcal{FM}$. First, suppose that P is a Hamiltonian path of G whose endvertices are u and v. Clearly, P is a spanning tree of G such that all endvertices of P are of degree at most 2 in G. By Lemma 1(iii), this implies $G \in \mathcal{FM}$. Next, suppose that $G \in \mathcal{FM}$. By Lemma 1(iii), this implies that G has a spanning tree T such that all endvertices of T are of degree at most 2 in G. Since u and v are the only vertices of G of degree at most 2, this implies that T has exactly the two endvertices u and v. Hence, T is a Hamiltonian path of G whose endvertices are u and v. $\square$

In order to enable suitable reductions, we now consider a slightly more general version of our decision problem.

Allowed Decycling Matching

Instance:	A graph G and a set F of edges of G.
Task:	Decide whether G has a decycling matching M that does not intersect F, and determine such a matching if it exists.

A matching M as in Allowed Decycling Matching is an *allowed decycling matching* of (G, F).

The *claw* $K_{1,3}$ and the *paw* $K_{1,3} + e$ are the unique graphs with degree sequences $1, 1, 1, 3$ and $1, 2, 2, 3$, respectively.

Theorem 3 Allowed Decycling Matching *can be solved in polynomial time for* $\{K_{1,3}, K_{1,3}+e\}$*-free graphs.*

Proof: Let G be a $\{K_{1,3}, K_{1,3} + e\}$-free graph and let F be a set of edges of G. Since (G, F) has an allowed decycling matching if and only if $(K, E(K) \cap F)$ has an allowed decycling matching for every component K of G, we may assume that G is connected.

The following claim is an immediate consequence of Lemma 1(ii).

Claim 1 *If G contains K_4 as an induced subgraph, then (G, F) has no allowed decycling matching.*

Claim 2 *If G has a vertex of degree at least 4, then (G, F) has no allowed decycling matching.*

Proof of Claim 2: Let u be a vertex of G with four neighbors v_1, v_2, v_3, and v_4. Since G is $\{K_{1,3}, K_{1,3}+e, K_4\}$-free, we may assume, by symmetry, that $v_1v_2, v_2v_3 \in E(G)$ and $v_1v_3 \notin E(G)$. Considering v_1, v_3, and v_4, this implies, by symmetry, that $v_3v_4 \in E(G)$. Considering the three triangles uv_1v_2u, uv_2v_3u, and uv_3v_4u, it follows that (G, F) does not have an allowed decycling matching. $\square$

Since no endvertex of G lies on a cycle, we may assume that G has minimum degree at least 2. Since whether G is K_4-free and has maximum degree at most 3, can be tested in polynomial time, we may assume, by Claim 1 and Claim 2, that G is K_4-free and has maximum degree at most 3. If G does not have any vertex of degree 3, then G is a cycle, and (G, F) has an allowed decycling matching if and only if F does not contain all edges of G. Hence, we may assume that G has a vertex b of degree 3. Let $N_G(b) = \{a, c, d\}$. Since G is $\{K_{1,3}, K_{1,3} + e, K_4\}$-free, we may assume, by symmetry, that $ac, cd \in E(G)$ and $ad \notin E(G)$. Let $G' = (V(G) \setminus \{b, c\}, (E(G) \setminus \{ab, ac, bc, bd, cd\}) \cup \{ad\})$, and let $F' = (F \setminus \{ab, ac, bc, bd, cd\}) \cup \{ad\} \cup \{xa : x \in N_G(a) \setminus \{b, c\}\} \cup \{ad\} \cup \{yd : x \in N_G(d) \setminus \{b, c\}\}$.

Claim 3 *(i) G' is $\{K_{1,3}, K_{1,3} + e\}$-free.*

(ii) (G, F) has an allowed decycling matching if and only if

- *$ab, cd \notin F$ or $ac, bd \notin F$, and*
- *(G', F') has an allowed decycling matching.*

Proof of Claim 3: (i) Suppose, for a contradiction, that G' contains an induced subgraph H that is isomorphic to $K_{1,3}$ or $K_{1,3}+e$. Since G is $\{K_{1,3}, K_{1,3}+e\}$-free, H contains the edge ad. By symmetry, we may assume that either $d_H(a) = 3$ or $d_H(a) = d_H(d) = 2$. If $d_H(a) = 3$, then $G[(V(H) \setminus \{d\}) \cup \{b\}]$ is isomorphic to $K_{1,3}$ or $K_{1,3}+e$, which is a contradiction. If $d_H(a) = d_H(d) = 2$, and x is the common neighbor of a and d in H, then $G[\{a, b, c, x\}]$ is isomorphic to $K_{1,3} + e$, which is a contradiction.

(ii) First, we assume that M is an allowed decycling matching of (G, F). Since $G[\{a, b, c, d\}] - (M \cap E(G[\{a, b, c, d\}]))$ is a forest, we obtain that $M \cap E(G[\{a, b, c, d\}])$ is either $\{ab, cd\}$ or $\{ac, bd\}$, that is, $ab, cd \notin F$ or $ac, bd \notin F$. Since $G - M$ contains either the path $abcd$ or the path $acbd$, it follows that $M \setminus E(G[\{a, b, c, d\}])$ is an allowed decycling matching of (G', F').

Next, we assume that M' is an allowed decycling matching of (G', F'), and that, by symmetry, $ab, cd \notin F$. Since $ad \in F'$, we obtain that $ad \notin M'$. This implies that $M' \cup \{ab, cd\}$ is an allowed decycling matching of (G, F). □

Iteratively applying the reductions captured by the above claims, clearly allows to decide in polynomial time whether (G, F) has an allowed decycling matching, and to determine such a matching in polynomial time if it exists. □

Theorem 4 Allowed Decycling Matching *can be solved in polynomial time for P_5-free graphs.*

Proof: Let G be a P_5-free graph and let F be a set of edges of G. Again, we may assume that G is connected. By a result of Liu et al. [7], G has a dominating induced cycle of length 5 or a dominating clique.

First, we assume that G has a dominating induced cycle C of length 5. Since G is P_5-free, every vertex in $V(G) \setminus V(C)$ has at least two neighbors in $V(C)$. This implies that $m(G) \geq 5+2(n(G)-5) = 2n(G) - 5$. If $n(G) \geq 9$, then $m(G) > \frac{3n(G)}{2} - 1$, and Lemma 1(ii) implies that (G, F) has no allowed decycling matching. Hence, we may assume that $n(G) \leq 8$ in this case, which implies that Allowed Decycling Matching can be solved in constant time.

Next, we assume that G has a dominating clique C of order p. Lemma 1(ii), we may assume that $p \leq 3$. If $p = 1$, then G has a universal vertex u. Now, it follows easily that (G, F) has an allowed decycling matching if and only if one of the following two conditions holds:

- $G - u$ is a graph of maximum degree at most 1 that contains no edge from F.
- u has a neighbor v of degree at most 2 such that $uv \notin F$, and $G - \{u, v\}$ is a graph of maximum degree at most 1 that contains no edge from F.

The first condition is equivalent to the existence of an allowed decycling matching that contains no edge incident with u, while the second condition is equivalent to the existence of an allowed decycling matching that contains the edge uv.

If $p = 3$, then G has a dominating triangle $uvwu$. Clearly, every decycling matching must contain one of the three edges of $uvwu$. Furthermore, u, v, and w belong to the same component of $G - M$ for every matching M of G. Considering the polynomially many possibilities to select one of the three edges of $uvwu$ as well as at most one further edge incident to a vertex in C, both forming a matching that does not intersect F, we can deduce a polynomial number of efficiently checkable conditions, very similar to those explicitly stated in the case $p = 1$, such that (G, F) has an allowed decycling matching if and only if one of these conditions is satisfied.

Finally, if $p = 2$, then we may assume that no vertex in $V(G) \setminus C$ has two neighbors in C; since otherwise, G has a dominating triangle. This implies that if uv is a dominating edge of G, then $V(G)$ is partitioned into $\{u, v\}$, $N_G(u) \setminus \{v\}$, and $N_G(v) \setminus \{u\}$. Clearly, every matching contains at most two edges incident with u and v. Again, considering the polynomially many possibilities to select such edges, we can deduce a polynomial number of efficiently checkable conditions such that (G, F) has an allowed decycling matching if and only if one of these conditions is satisfied. For example, (G, F) has an allowed decycling matching that contains uv if and only if $uv \not\in F$, and there is a set E containing at most one edge between $N_G(u) \setminus \{v\}$ and $N_G(v) \setminus \{u\}$ such that $(G - \{u, v\}) - E$ has maximum degree at most 1, and contains no edge from F. Clearly, this implies the desired statement. $\square$

Theorem 5 Allowed Decycling Matching *can be solved in polynomial time for chordal graphs.*

Proof: Let G be a chordal graph and let F be a set of edges of G. Again, we may assume that G is connected. Furthermore, we may assume that G does not contain bridges, that is, every block of G has order at least 3. If G contains a cycle C of length ℓ at least 5, then, since G is chordal, $m(G[V(C)]) \geq 2\ell - 3 > \frac{3\ell}{2} - 1$, and Lemma 1(ii) implies that G does not have an allowed decycling matching. Hence, every cycle of G has length at most 4.

Let B be a block of G of order at least 4. Since G is chordal, B contains a triangle $abca$. Since B has order at least 4, we may assume that b has a neighbor d distinct from a and c. Since G has no cycle of length at least 5, considering a shortest path in $B - b$ between d and a vertex in $\{a, c\}$ implies that we may assume, by symmetry, that c and d are adjacent. By Lemma 1(ii), G does not contains K_4, which implies that $G[\{a, b, c, d\}]$ is $K_4 - e$. Suppose that B has order at least 5. This implies that some vertex x in $V(B) \setminus \{a, b, c, d\}$ has a neighbor y in $\{a, b, c, d\}$. Considering a shortest path in $B - y$ between x and a vertex in $\{a, b, c, d\} \setminus \{y\}$, and using the absence of cycles of length at least 5, we obtain that x is adjacent to a and d. Hence, $m(G[\{a, b, c, d, x\}]) \geq 7 > \frac{3 \cdot 5}{2} - 1$, and Lemma 1(ii) implies that G does not have an allowed decycling matching. Hence, we may assume that every block of G is either a triangle or isomorphic to $K_4 - e$.

Clearly, we may assume that G has at least 2 blocks. Let B be an endblock of G, that is, B contains exactly one cutvertex.

First, we assume that B is a triangle $abca$ and that c is the cutvertex of B. If $ab, bc, ca \in F$, then (G, F) does not have an allowed decycling matching. If $ab \not\in F$, then (G, F) has an allowed decycling matching if and only if $(G - \{a, b\}, F \setminus \{ab, bc, ca\})$ has an allowed decycling matching. If $ab \in F$ but $ac \not\in F$ or $bc \not\in F$, then (G, F) has an allowed decycling matching if and only if $\big(G - \{a, b\}, (F \setminus \{ab, bc, ca\}) \cup \{cx : x \in N_G(c) \setminus \{a, b\}\}\big)$ has an allowed decycling matching.

Next, we assume that B is isomorphic to $K_4 - e$. Let x denote the cutvertex contained in B. It follows that (G, F) has an allowed decycling matching if and only if one of the two perfect matchings of B does not intersect F, and $\big(G - (V(B) \setminus \{x\}), (F \setminus E(B)) \cup \{xy : y \in N_G(x) \setminus V(B)\}\big)$ has an allowed decycling matching.

Iteratively applying these reductions allows to decide in polynomial time whether (G, F) has an allowed decycling matching, and also, to determine such a matching in polynomial time if it exits. $\square$

Theorem 6 Allowed Decycling Matching *can be solved in polynomial time for C_4-free distance hereditary graphs.*

Proof: Let G be a (connected) C_4-free distance hereditary graph and let F be a set of edges of G. Clearly, we may assume that G has order at least 3. By results of Bandelt and Mulder [1], this implies that G has an endvertex, or that G contains two vertices a and b with either $N_G[a] = N_G[b]$ or $N_G(a) = N_G(b)$. Clearly, if G has an endvertex x, then (G, F) has an allowed decycling matching if and only if $(G - x, F \cap E(G - x))$ has an allowed decycling matching. Hence, we may assume that G has no endvertex, and that the second case occurs. If $|N_G(a) \setminus \{b\}| \geq 3$, then, since G is C_4-free, we obtain $m(G[N_G[a]]) > \frac{3}{2}|N_G[a]| - 1$, and Lemma 1(ii) implies that (G, F) does not have an allowed decycling matching. If $|N_G(a) \setminus \{b\}| = 2$, then, since G is C_4-free, $G[N_G[a]]$ is isomorphic either to K_4 or to $K_4 - e$. In the first case, Lemma 1(ii) implies that (G, F) does not have an allowed decycling matching, and, in the second case, (G, F) has an allowed decycling matching, if and only if some perfect matching of $G[N_G[a]]$ does not intersect F, and $(G - \{a, b\}, (F \cap E(G - \{a, b\})) \cup \{uv \in E(G - \{a, b\}) : u \in N_G[a] \setminus \{a, b\}\})$ has an allowed decycling matching. If $|N_G(a) \setminus \{b\}| = 1$, then, since G does not have an endvertex, a and b are adjacent and lie on a triangle with the unique vertex c in $N_G(a) \cap N_G(b)$. If $ab, bc, ca \in F$, then (G, F) does not have an allowed decycling matching, if $ab \notin F$, then (G, F) has an allowed decycling matching if and only if $(G - \{a, b\}, F \cap E(G - \{a, b\}))$ has an allowed decycling matching, and if $ab \in F$ and $ac \notin F$, then (G, F) has an allowed decycling matching if and only if $(G - \{a, b\}, (F \cap E(G - \{a, b\})) \cup \{uc \in E(G) : u \in N_G(c) \setminus \{a, b\}\})$ has an allowed decycling matching. Iteratively applying these reductions allows to decide in polynomial time whether (G, F) has an allowed decycling matching, and also, to determine such a matching in polynomial time if it exits. $\Box$

3 Conclusion

As observed in the introduction, we only study a special case of the more general problem of destroying all cycles by removing edges under the restriction that the graph formed by the removed edges has bounded maximum degree. This problem can certainly be considered more generally. Furthermore, one can consider variants, such as, for instance, deciding whether a given graph is the union of a bipartite graph and a matching, that is, whether it has a matching whose removal destroys all odd cycles. Another class of graphs where Allowed Decycling Matching might be solvable efficiently are chordal bipartite graphs. Their guaranteed density is close to the threshold from Lemma 1(ii), which should imply strong restrictions on the block structure of chordal bipartite graphs with a decycling matching.

References

[1] H.-J. Bandelt, H.M. Mulder, Distance-hereditary graphs, Journal of Combinatorial Theory, Series B 41 (1986) 182-208.

[2] S. Bau, L.W. Beineke, The decycling number of graphs, Australasian Journal of Combinatorics 25 (2002) 285-298.

[3] L.W. Beineke, R.C. Vandell, Decycling graphs, Journal of Graph Theory 25 (1997) 59-77.

[4] P. Erdős, M. Saks, V.T. Sós, Maximum induced trees in graphs, Journal Combinatorial Theory, Series B 41 (1986) 61-79.

[5] M.R. Garey, D.S. Johnson, R.E. Tarjan, The planar Hamiltonian circuit problem is NP-complete, SIAM Journal on Computing 5 (1976) 704-714.

[6] M. Gentner, D. Rautenbach, Feedback vertex sets in cubic multigraphs, Discrete Mathematics 338 (2015) 2179-2185.

[7] J. Liu, H. Zhou, Dominating subgraphs in graphs with some forbidden structures, Discrete Mathematics 135 (1994) 163-168.

[8] D. Rautenbach, Dominating and large induced trees in regular graphs, Discrete Mathematics 307 (2007) 3177-3186.

[9] D.J. Rose, R.E. Tarjan, G.S. Lueker, Algorithmic aspects of vertex elimination on graphs, SIAM Journal on Computing 5 (1975) 266-283.

Appendix G:

Odd Decycling with a matching

This appendix contains the extended abstract "*Odd Decycling with a matching*" submitted on February to *43rd International Workshop on Graph-Theoretic Concepts in Computer Science* (WG 2017).

Eliminating Odd Cycles by Removing a Matching

Carlos V.G.C. Lima[1], Dieter Rautenbach[2], Uéverton S. Souza[3], and Jayme L. Szwarcfiter[1]

[1] PESC, COPPE, Federal University of Rio de Janeiro, Rio de Janeiro, Brazil.
gclima@cos.ufrj.br, jayme@nce.ufrj.br

[2] Institute of Optimization and Operations Research, Ulm University, Ulm, Germany.
dieter.rautenbach@uni-ulm.de

[3] Institute of Computing, Fluminense Federal University, Niterói, Brazil.
ueverton@ic.uff.br

Abstract. We study the problem of determining whether a given graph $G = (V, E)$ admits a matching M whose removal destroys all odd cycles of G (or equivalently whether $G - M$ is bipartite). This problem is also equivalent to determine whether G admits an $(1,1)$-coloring, which is a 2-coloring of $V(G)$ in which each color class induces a graph of maximum degree at most 1. We found a dichotomy related to the NP-completeness such a decision problem, we show that ODD DECYCLING MATCHING is NP-complete even for 3-colored planar graphs of maximum degree 4, but can be solved in linear time in graphs of maximum degree 3. In addition, we present a polynomial time algorithm for the case where the input graph G contains only triangles as odd cycles. Polynomial time algorithms for (claw, paw)-free graphs, graphs with bounded dominating sets, and P_5-free graphs are also presented. Additionally, we show that the problem is fixed-parameter tractable when parameterized by clique-width, which implies polynomial time solvability for many interesting graph classes such as distance-hereditary graphs and outerplanar graphs.

Keywords: Odd decycling matching, $(1,1)$-coloring, edge-deletion, bipartization, frustration, planar graphs.

1 Introduction

Given a graph $G = (V, E)$ and a graph property Π, the Π *edge-deletion problem* consists in determining the minimum number of edges required to be removed in order to obtain a graph satisfying Π [9]. Given an integer $k \geq 0$, the Π *edge-deletion decision problem* asks for a set $F \subseteq E(G)$ with $|F| \leq k$, such that the obtained graph by the removal of F satisfies Π. Both versions have received widely attention on the study of their complexity, where we can cite [2, 9, 19, 22, 28, 31, 32] and references therein for applications. When the obtained graph is required to be bipartite, the corresponding edge- (vertex-) deletion problem is called *edge* (*vertex*) *bipartization* [1, 11, 18] or *edge* (*vertex*) *frustration* [33]. Choi, Nakajima, and Rim [11] showed that the edge bipartization decision problem is NP-complete even for cubic graphs.

Furmańczyk, Kubale, and Radziszowski [18] considered vertex bipartization of cubic graphs by the removal of an independent set. In this paper we study the analogous edge deletion decision problem, that is, the problem of determining whether a finite, simple, and undirected graph G admits a removal of a set of edges that is a matching in G in order to obtain a bipartite graph. Formally, for a set M of edges of a graph $G = (V,E)$, let $G - M$ be the graph with vertex set $V(G)$ and edge set $E(G) \setminus M$. For a matching $M \subseteq E(G)$, we say that M is an *odd decycling matching* of G if $G - M$ is bipartite. Let $\mathscr{BM}$ denote the set of all graphs admitting an odd decycling matching. We deal with the complexity of the following decision problem.

ODD DECYCLING MATCHING
Input: A finite, simple, and undirected graph G.
Question: Does $G \in \mathscr{BM}$?

A more restricted version of this problem is considered by Schaefer [29]. He deals with the problem of determining whether a given graph G admits a 2-coloring of the vertices so that each vertex has *exactly* one neighbor with same color as itself. We can see that the removal of the set of edges whose endvertices have same color, which is a perfect matching of G, generates a bipartite graph. Schaefer proved that such a problem is NP-complete even for planar cubic graphs.

With respect to the minimization version, the edge-deletion decision problem in order to obtain a bipartite graph is analogous to SIMPLE MAX CUT, which was proved to be NP-complete by Garey, Johnson and Stockmeyer [19]. Yannakakis [31] proved its NP-completeness even for cubic graphs.

ODD DECYCLING MATCHING can also be seen as another problem. A graph G is $(d_1,\dots,d_k)$*-colorable* if $V(G)$ can be partitioned into $V_1,\dots,V_k$, such that the induced subgraph $G[V_i]$ has maximum degree at most d_i, for all $1 \le i \le k$. This is a generalization of the classical *proper k-coloring*, when every $d_i = 0$, and the *d-improper k-coloring*, when every $d_i = d \ge 1$. It is clear to see that $G \in \mathscr{BM}$ if and only if G is $(1,1)$-colorable. Lovász [25] proved that if a graph G satisfies $(d_1+1)+(d_2+1)+\cdots+(d_k+1) \ge \Delta(G)+1$ then G is $(d_1,\dots,d_k)$-colorable, where $\Delta(G)$ denotes the *maximum degree* of G. This result shows that every subcubic graph is $(1,1)$-colorable and thus belongs to $\mathscr{BM}$. Borodin, Kostochka, and Yancey [6] studied the $(1,1)$-colorable graphs with respect to the sparseness parameter $mad(G) = \max\left\{\frac{2|E(H)|}{|V(H)|}, \text{for all } H \subseteq G\right\}$. They proved that every graph G with $mad(G) \le \frac{14}{5}$ is $(1,1)$-colorable, where this bound is sharp. Moreover, they defined the parameter $\rho(G) = \min_{S \subseteq V(G)} \rho_G(S)$, such that $\rho_G(S) = 7|S| - 5|E(G[S])|$. They showed that G is $(1,1)$-colorable if $\rho(G) \ge 0$. Finally, they also proved that every planar graph with *girth* (the size of the smallest cycle of G) at least 7 is $(1,1)$-colorable. This is the best result concerning $(1,1)$-coloring of planar graphs.

In this work we summarize our results as follows. We prove that ODD DECYCLING MATCHING is NP-complete even for planar graphs with maximum degree 4. As positive results, we show polynomial time algorithms for (claw, paw)-free graphs, graphs that have only triangles as odd cycles, and graphs that have a small dominating set. We also show that graphs in $\mathscr{BM}$ can be expressed in monadic second order logic. Hence, using MSOL's meta-theorems [12, 13, 15, 16] we prove that ODD DECYCLING MATCHING is fixed-parameter tractable when parameterized by clique-width. We also show a exact $2^{O(vc(G))}.n$ algorithm, where $vc(G)$ is the *vertex cover number* of G. Finally, for a generalization of ODD DECYCLING MATCHING, we show a kernel with at most $2.nd(G)$ vertices when such a more general problem is parameterized by *neighborhood diversity number*, $nd(G)$.

1.1 Preliminaries

A *diamond* is the graph obtained by removing one edge from the K_4. Let W_k be the *wheel graph* of order k, that is, the graph containing a vertex v, called *central*, and a cycle C of order k, such that v is adjacent to all vertices of C. We say that a graph is a *k-pool* if it is formed by k triangles edge disjoint whose bases induce a C_k. Formally, a k-pool is obtained from a cycle $C = \{v_1, v_2, \dots, v_{2k}\}$ $(k \geq 3)$, such that the odd-indexed vertices induce a cycle $p_1 p_2 \dots p_k p_1$, called *internal cycle* of the k-pool, where $p_i = v_{2i-1}$, $1 \leq i \leq k$. The even-indexed vertex b_i is the i-th-*border* of the k-pool, where $\{b_i\} = N_C(p_i) \cap N_C(p_{i+1})$ and $i+1$ is taken modulo $2k$. The *claw* $(K_{1,3})$ and the *paw* graphs are the unique with degree sequences $1,1,1,3$ and $1,2,2,3$, respectively.

Clearly, every graph $G \in \mathscr{BM}$ admits a proper 4-coloring. Hence every graph in $\mathscr{BM}$ is K_5-free. More precisely, every graph in $\mathscr{BM}$ is W_4-free. Hence some proper 4-colorable graphs do not admits an odd decycling matching. Lemma 1 collects some properties of graphs in $\mathscr{BM}$.

Lemma 1. *Given a graph G in $\mathscr{BM}$ and an odd decycling matching M of G, the following assertions are true.*

(i) *If G has a diamond D as a subgraph, then M contains no edge $e \notin E(D)$ incident to only one vertex of degree three of D.*
(ii) *$G[N_G(v)]$ cannot contain two disjoint P_3, for every $v \in V(G)$.*
(iii) *G cannot contain a W_k as a subgraph, for all $k \geq 4$.*
(iv) *G cannot contain a k-pool as a subgraph, for all odd $k \geq 3$.*

Bondy and Locke [5] presented the following lemma, which was also obtained by Erdős [17] by induction on $n(G)$.

Lemma 2. *(Bondy and Locke [5]) Let G be a graph and let B be a largest bipartite subgraph of G. Then $d_B(v) \geq \frac{1}{2} d_G(v)$, for every $v \in V(G)$.*

Lemma 2 shows that every subcubic graph G admits an odd decycling matching, since every vertex has at most one incident edge not in a largest bipartite subgraph of G. This result was also obtained by Lovász [25] with respect to 1-improper 2-coloring of graphs with maximum degree at most 3.

Consider a bipartition of $V(G)$ into sets A and B. For every vertex v, we say that v is of type (a,b) if $d_{V(G)\setminus X}(v) = a$ and $d_X(v) = b$, where X is the part (either A or B) which contains v. We present a linear algorithm to find an odd decycling matching of subcubic graphs, Algorithm 1.

Theorem 1. *Algorithm 1 returns in linear time an odd decycling matching for subcubic graphs.*

Despite the simplicity of Algorithm 1, determining the size of a minimum odd decycling matching of subcubic graphs is NP-hard, since this problem becomes analogous to MAX CUT [20] for such a class.

Algorithm 1: A linear time algorithm that determines an odd decycling matching for subcubic graphs.

```
Data: A subcubic graph $G$.
1  $A \leftarrow$ A maximal independent set of $G$;
2  $B \leftarrow V(G) \setminus A$;
3  $M \leftarrow \emptyset$;
4  while exists a vertex $v \in B$ of type $(1,2)$, with respect to $A$ and $B$, do
5      $u \leftarrow N_{G[A]}(v)$;
6      if $u$ is of type $(3,0)$ then
7          $B \leftarrow B \setminus \{v\}$;
8          $A \leftarrow A \cup \{v\}$;
9      else
10         $B \leftarrow \{B \cup \{u\}\} \setminus \{v\}$;
11         $A \leftarrow \{A \cup \{v\}\} \setminus \{u\}$;
12 $M \leftarrow$ all edges of $G[A] \cup G[B]$;
13 return $M$;
```

2 NP-Completeness for ODD DECYCLING MATCHING

In this section we prove that ODD DECYCLING MATCHING is NP-complete even for planar graphs of maximum degree at most 4.

We consider a well-known NP-complete problem so-called POSITIVE PLANAR 1-IN-3-SAT [27]. In order to prove the NP-completeness of ODD DECYCLING MATCHING, we must first observe that there is a polynomial time reduction from POSITIVE PLANAR 1-IN-3-SAT to the following decision problem:

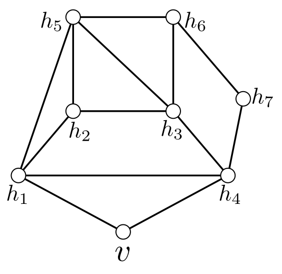
- PLANAR 1-IN-3-SAT$_3$: Version of PLANAR 1-IN-3-SAT where each clause has either 2 or 3 literals and each variable occurs at most 3 times. Moreover, each positive literal occurs at most twice, while every negative literal occurs at most once in F.

Theorem 2. PLANAR 1-IN-3-SAT$_3$ *is NP-complete.*

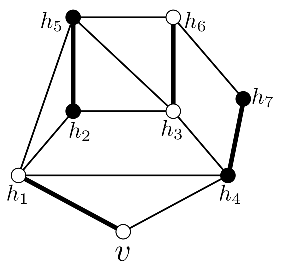
The next simple lemma is used in the correctness of our reduction. Let us call the graph depicted in Fig 1a by *head*. Vertex v is the *neck* of the head.

Lemma 3. *Let G be a graph that contains an induced subgraph H isomorphic to a head graph, whose neck is v. Then all edges not in H incident to v cannot be in any odd decycling matching of G. Moreover H admits only one odd decycling matching.*

With Lemma 3 we can establish the NP-completeness of ODD DECYCLING MATCHING. Remember that graphs in $\mathcal{BM}$ are all 4-colorable. Moreover planar graphs are classical 4-colorable graphs. Hence it is interesting to know what happens in such a class. The next result shows that the NP-completeness is also obtained even for 3-colorable planar and bounded degree graphs. The circles with an H in the figures represent an induced subgraph isomorphic to the head graph, whose neck is the vertex touching the circle. By simplicity, this pattern will be used in the remaining figures whenever possible.



(a) The head graph H.



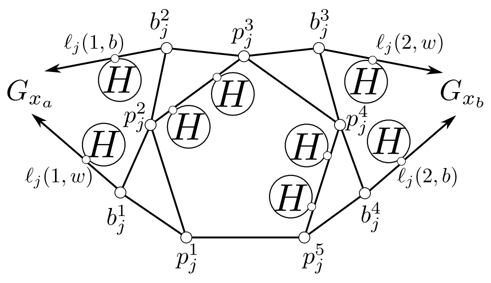
(b) The odd decycling matching of H.

Fig. 1: The head graph and its odd decycling matching M.

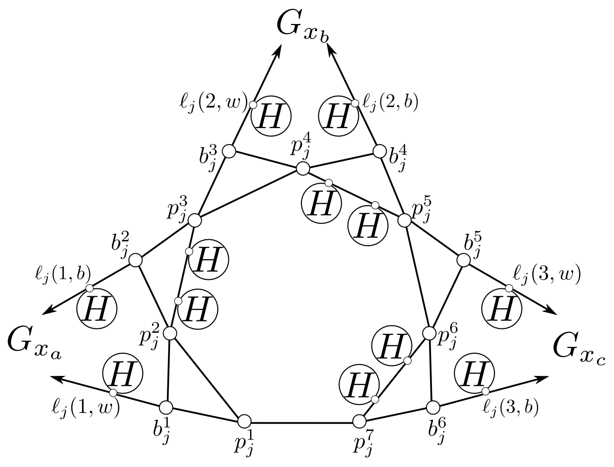
2.1 *NP*-Completeness for Planar Graphs with Maximum Degree at Most 4

We prove the *NP*-completeness by a reduction from PLANAR 1-IN-3-SAT$_3$. In order to prove this result, next we give a useful lemma.

Lemma 4. *Let b be a border of an odd k-pool graph G, such that c_1 and c_k are its neighbors in G. It follows that every odd decycling matching of $G - b$ must contain exactly one edge of the internal cycle, which is different from $c_1 c_k$. Moreover, there is only one odd decycling matching for each such an edge.*



(a) For clauses of size two.



(b) For clauses of size three.

Fig. 2: Clause gadget G_{c_j} in Theorem 3. Fig. 2a is such that $c_j = (x_a \vee x_b)$. Fig. 2b is such that $c_j = (x_a \vee x_b \vee x_c)$.

Theorem 3. ODD DECYCLING MATCHING *is NP-complete even for* 3*-colorable planar graphs with maximum degree at most* 5.

Proof. Let F be an instance of PLANAR 1-IN-3-SAT$_3$, with $X = \{x_1, x_2, \dots, x_n\}$ and $C = \{c_1, c_2, \dots, c_m\}$ be the sets of variables and clauses of F, respectively. We construct a planar graph $G = (V, E)$ of maximum degree 5 as follows:

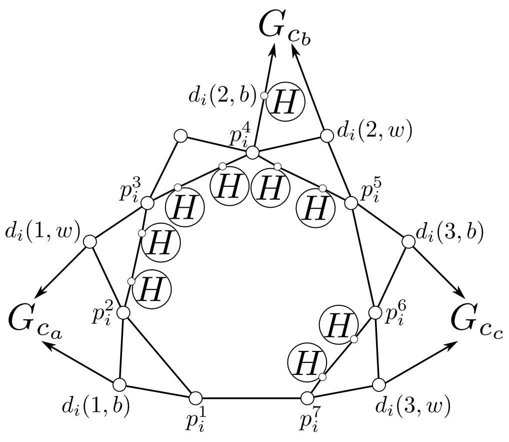


Fig. 3: Variable gadget G_{x_i} in Theorem 3. Each pair of edges with one no endvertex connects G_{x_i} to one clause gadget G_{c_a}, G_{c_b}, or G_{c_c}, where $x_i \in (c_a \cap c_b \cap c_c)$.

- For each clause $c_j \in C$, we construct a gadget G_{c_j} as depicted in Fig. 2. Such gadgets are just a 5-pool and a 7-pool less a border for clauses of size 2 and 3, respectively. Moreover, for the alternate edges of the internal cycle we subdivide them twice and append a head graph to each such a new vertex. Finally, we add two vertices $\ell_j(k,w)$ and $\ell_j(k,b)$, such that $b_j^{2k-1}\ell_j(k,w) \in E(G)$ and $b_j^{2k}\ell_j(k,b) \in E(G)$, for $k \in \{1,2,3\}$. For such new vertices, we append a head graph to each one.
- For each variable $x_i \in X$, we construct a gadget G_{x_i} as depicted in Fig. 3. Such a gadget is a 7-pool less a border, where we subdivide the edges $p_i^2p_i^3$, $p_i^3p_i^4$, $p_i^4p_i^5$, and $p_i^6p_i^7$ twice, where every such a new vertex has a pendant head. We rename each border vertex b_i^{2k-1} $(k \in \{1,3\})$ as $d_i(k,b)$ and b_i^{2k} as $d_i(k,w)$, for $k \in \{1,2,3\}$. Moreover we add a new vertex $d_i(2,b)$ adjacent to p_i^4, which has a pendant head graph.
- The connection between clause and variable gadgets are as in Fig. 2 and Fig. 3, where each pair of arrow head edges in a variable gadget G_{x_i} corresponds to a pair of such edges in a clause gadget G_{c_j}, such that $x_i \in c_j$. More precisely, if $x_i \in c_j$, then we add the edges $\ell_j(k,b)d_i(k',b)$ and $\ell_j(k,w)d_i(k',w)$, for some $k \in \{1,2,3\}$ and for some $k' \in \{1,2\}$. On the other hand, if $\overline{x_i} \in c_j$, then we add the edges $\ell_j(k,b)d_i(3,b)$ and $\ell_j(k,w)d_i(3,w)$, for some $k \in \{1,2,3\}$.
- If a variable occurs only twice in F, then just consider those connections corresponding to the literals of x_i in the clauses of F, such that $d_i(3,b)$ and $d_i(3,w)$ represent $\overline{x_i}$.

Let G be the graph obtained from F by the above construction. We can see that G has maximum degree 5, where the only vertices with degree 5 are those p_i^4, for each variable gadget G_{x_i}. Furthermore, it is clear that G is 3-colorable.

It remains to show that if F is planar (that is, if G_F is planar), then G is planar. Consider a planar embedding ψ of G_F. We replace each variable vertex v_{x_i} of G_F by a variable gadget G_{x_i}, as well as every clause vertex v_{c_j} of G_{c_j} by a clause gadget G_{c_j}. The clause gadgets correspond to clauses of length two or three, which depends on the degree of v_{c_j} in G_F. Since the clause and variable gadgets are planar, we just need to show that the connections among them keep the planarity. Given an edge $v_{x_i}v_{c_j} \in E(G_F)$, we connect G_{x_i} and G_{c_j} by duplicating such an edge as parallel edges $\ell_j(k,w)\ell_i(k',w)$ and $\ell_j(k,b)\ell_i(k',b)$, for some $k \in \{1,2,3\}$ and for some $k' \in \{1,2\}$ or $\ell_j(k,b)\ell_i(3,b)$ and $\ell_j(k,w)\ell_i(3,w)$, for some $k \in \{1,2,3\}$, as previously discussed.

In order to prove that F is satisfiable if and only if $G \in \mathscr{BM}$, we discuss some considerations related to odd decycling matchings of the clause and variable gadgets. By Lemma 4, we know that an odd k-pool graph less a border admits one odd decycling matching for each edge of the internal cycle, except that whose both endvertices are adjacent to the removed border. Furthermore, by Lemma 3 it follows that each external edge incident to the neck of an induced head cannot be in any odd decycling matching. In this way, Fig. 4 shows the possible odd decycling matchings M, given by the stressed edges for the clause gadget G_{c_j} of clauses of length three. The black and white vertex assignment represents the bipartition of $G_{c_j} - M$. Notice that exactly one pair of vertices $\ell_j(k,w)$ and $\ell_j(k,b)$ ($k \in \{1,2,3\}$) is such that they have the same color, while the other such pairs have opposite colors. More precisely, we can see that $\ell_j(k,w)$ has the same color for each pair with opposite color vertices as well as $\ell_j(k,b)$, for each odd decycling matching of G_{c_j}. In this way, we can associate one literal x_j^1, x_j^2, and x_j^3 to each pair of vertices $\ell_j(k,w)$ and $\ell_j(k,b)$, $k \in \{1,2,3\}$. A similar analysis can be done for clause gadgets of clauses of length two.

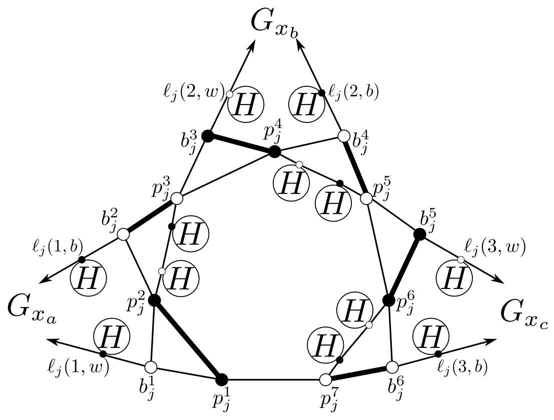
In the same fashion as the clause gadgets, each variable gadget G_{x_i} admits two possible odd decycling matchings M as depicted in Fig. 5. We can see that the pair $\ell_i(3,b)$ and $\ell_i(3,b)$ has a different assignment for the other two pairs $\ell_i(k,b)$ and $\ell_i(k,b)$, $k \in \{1,2\}$. Moreover, the last two pairs have the same assignment, as it can be seen in Fig. 5a and Fig. 5b. One more detail is that the unique possibilities for such pairs is such that $\ell_i(3,b)$ and $\ell_i(3,b)$ have opposite assignments if and only if the vertices $\ell_i(k,b)$ and $\ell_i(k,b)$ have the same assignment, $k \in \{1,2\}$. Therefore we can associate the positive literal x_i to the pairs $\ell_i(k,b)$ and $\ell_i(k,b)$, $k \in \{1,2\}$, while $\overline{x_i}$ can be represented by $\ell_i(3,b)$ and $\ell_i(3,b)$.

As observed above for clause gadgets, we can associate true value to the pair of vertices $\ell_j(k,w)$ and $\ell_j(k,b)$ with same color, $k \in \{1,2,3\}$. This implies that exactly one of them is true and, that is, exactly one literal of c_j has true value. Moreover, each variable gadget has two positive literals and a negative one, such that the positive and negative have opposite truth assignment.

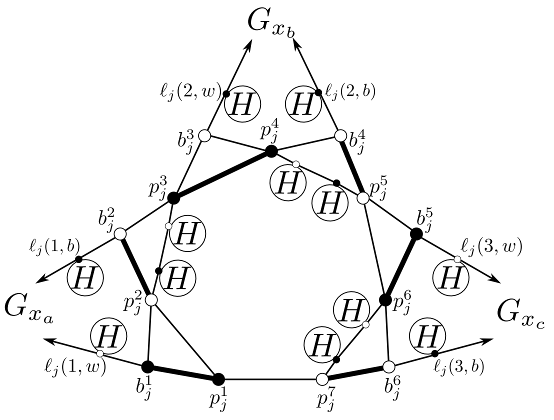
Hence, if $G \in \mathscr{BM}$, then every clause gadget has exactly one true literal and every variable has a correct truth assignment, which implies that F is satisfiable. Conversely, if F is satisfiable, then each clause has exactly one true literal. Thus, for each clause c_j gadget we associate to pair of vertices corresponding to its true literal a same color. By Fig.4, there is an appropriate choice of an odd decycling matching for each true literal of c_j. Moreover, for each literal gadget G_{x_i} there is also an appropriate odd decycling matching for the choice of the true literal. This concludes the proof. $\square$

By just slightly modifying the gadgets used in the previous proof, we obtain the following corollary.

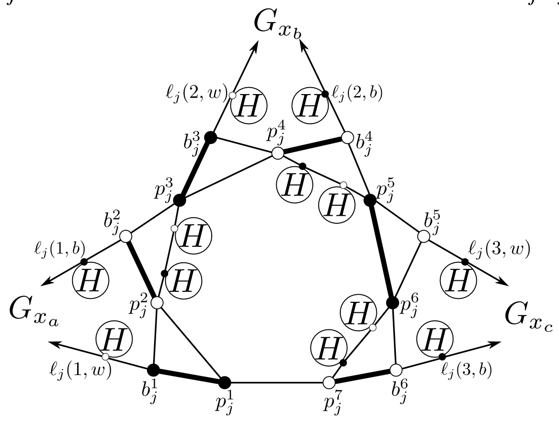
Corollary 1. ODD DECYCLING MATCHING *is NP-complete even for* 3*-colorable planar graphs with maximum degree* 4.



(a) $p_j^1 p_j^2 \in M$

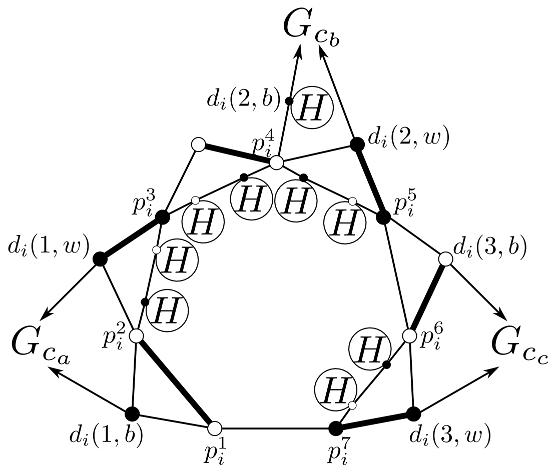


(b) $p_j^3 p_j^4 \in M$

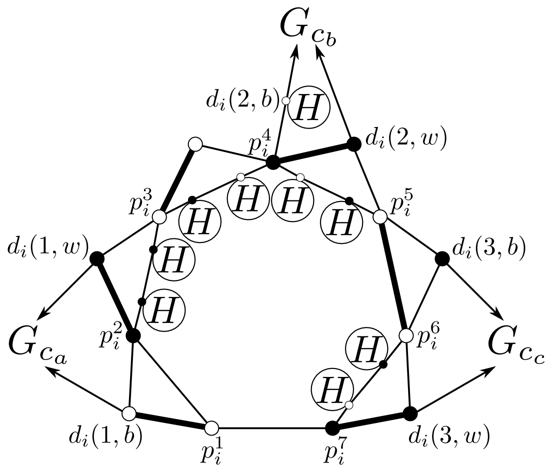


(c) $p_j^5 p_j^6 \in M$

Fig. 4: All possible configurations given by the removal of an odd decycling matching M, from a clause gadget G_{c_j} of three literals.



(a) $p_i^1 p_i^2 \in M$



(b) $p_i^5 p_i^6 \in M$

Fig. 5: All possible configurations given by the removal of an odd decycling matching M, from a variable gadget G_{x_i}.

3 Polynomial Time Results

In this section we present positive results about ODD DECYCLING MATCHING.

3.1 Graphs with Only Triangles as Odd Cycles

Given a graph G and a set F of edges of G, ALLOWED ODD DECYCLING MATCHING (AODM) consists of deciding whether G has an odd decycling matching M that does not intersect F, and determine such a matching if it exists. We may assume that G is connected.

Theorem 4. AODM *can be solved in polynomial time for graphs with only triangles as odd cycles.*

Proof. Let G be a graph without odd cycles of size at least 5 and let F be a set of edges of G. As described above, we can consider G connected. Moreover, we can assume G as a bridge-free graph, that is, a graph whose all blocks have size at least 3.

Consider a block decomposition Π of G. Clearly we can assume that G has at least two blocks. Let B be a block of G that contains exactly one cut-vertex v, that is, B is a final block in Π. If $G[B]$ is bipartite, then clearly (G,F) has an allowed odd decycling matching if and only if $(G^{'},F^{'})$ admits an allowed odd decycling matching, where $G^{'} = ((V(G)\setminus V(B))\cup\{v\},E(G)\setminus E(B))$ and $F^{'} = F\setminus E(B)$. Hence we can suppose that B has a triangle $v_1v_2v_3v_1$.

If $\{v_1v_2,v_1v_3\}$ is not a edge cut, then $G-\{v_1v_2,v_1v_3\}$ has a path P from v_1 to $\{v_2,v_3\}$. Consider P such a path of length at least 2 and such that it is a longest one. Moreover, consider v_2 as the first vertex reached by P between v_2 and v_3. In this way P must have the form v_1uv_2, otherwise either $G[V(P)\cup\{v_3\}]$ or $P\cup\{v_1v_2\}$ would contain an odd cycle of length at least 5, when P has either an even number of vertices or an odd number, respectively. It also follows that all path between v_1 and v_2, except v_1v_2, has length 2. Moreover, every path between v_1 and v_3, except v_1v_3, has length 2 and contains v_2.

If $uv_3\in E(G)$, then $N_B(v_1)=\{u,v_2,v_3\}$, otherwise let $w\in N_B(v_1)$. Since $d_B(w)\geq 2$, we get that $N_B(w)=\{v_1,v_2\}$, otherwise would exist a path between v_1 and v_2 of length at least 3, a contradiction. However the cycle $v_1uv_3v_2wv_1$ has length 5, a contradiction. Thus $G[\{v_1,v_2,v_3,u\}]$ is isomorphic to a K_4 and $\{v_1v_2,v_1v_3,v_1u\}$ is an edge cut. Hence B is a block of G and, by symmetry, $v=v_1$. In this case we get that (G,F) has an allowed odd decycling matching if and only if $(G^{'},F^{'})$ has an allowed odd decycling matching, where F does contain any matching of B of maximum size

$$G^{'} = ((V(G)\setminus V(B))\cup\{v\},E(G)\setminus E(B))$$

$$\text{and } F^{'} = (F\setminus E(B))\cup\{vx : x\in N_G(v)\setminus\{v_1,v_2,v_3\}\}.$$

If $uv_3\notin E(G)$, then $\{zv_2,zv_1\}$ is an edge cut for every vertex $z\in N_B(v_2)\cap N_B(v_1)$. Furthermore, by symmetry, $N_B(v_2)\cap N_B(v_1)$ is an independent set. Thus $\{v_1,v_2\}\cup\{N_B(v_2)\cap N_B(v_1)\}$ is a block of G. If $v\in\{v_1,v_2\}$ and $|N_B(v_2)\cap N_B(v_1)|\geq 3$, then (G,F)

has an allowed odd decycling matching is and only if $(G^{'}, F^{'})$ has an allowed odd decycling matching, where $v_1v_2 \notin F$ and

$$G^{'} = ((V(G) \setminus V(B)) \cup \{v\}, E(G) \setminus E(B)) \text{ and } F^{'} = (F \setminus E(B)) \cup \{vx : x \in N_G(v) \setminus V(B)\}.$$

If $v \in \{v_1, v_2\}$ and $|N_B(v_2) \cap N_B(v_1)| = 2$, then (G,F) has an allowed odd decycling matching if and only if $(G^{'}, F^{'})$ has an allowed odd decycling matching, where $v_1v_2 \notin F$, or $F \not\supseteq \{v_1u, v_2v_3\}$, or $F \not\supseteq \{v_1v_3, v_2u\}$, and

$$G^{'} = ((V(G) \setminus V(B)) \cup \{v\}, E(G) \setminus E(B)), \text{ and } F^{'} = (F \setminus E(B)) \cup \{vx : x \in N_G(v) \setminus V(B)\},$$

If $v \notin \{v_1, v_2\}$, then (G,F) has an allowed odd decycling matching if and only if $(G^{'}, F^{'})$ has an allowed odd decycling matching, where $v_1v_2 \notin F$ and

$$G^{'} = ((V(G) \setminus V(B)) \cup \{v\}, E(G) \setminus E(B)) \text{ and } F^{'} = (F \setminus E(B)).$$

Finally it remains the case that $G[\{v_1, v_2, v_3\}]$ is a block and, by symmetry, suppose $v = v_1$. Thus (G,F) has an allowed odd decycling matching if and only if $(G^{'}, F^{'})$ has an allowed odd decycling matching, where $v_2v_3 \notin F$ and

$$G^{'} = ((V(G) \setminus V(B)) \cup \{v\}, E(G) \setminus E(B)) \text{ and } F^{'} = F \setminus E(B),$$

or $v_1v_2 \notin F$ or, $v_1v_3 \notin F$, and

$$G^{'} = ((V(G) \setminus V(B)) \cup \{v\}, E(G) \setminus E(B)), \text{ and } F^{'} = (F \setminus E(B)) \cup \{vx : x \in N_G(v) \setminus V(B)\}.$$

This concludes the proof. □

3.2 Graphs with Small Dominating Sets

We will show now that ODD DECYCLING MATCHING can be solved in polynomial time when the given graph G has a dominating set of constant size. Such a result generalizes some known graph classes, as fo example P_5-free graphs [10], since the graphs in $\mathscr{BM}$ do not admit K_5 as subgraph.

Theorem 5. *Let k be a positive integer. For a graph G whose domination number is at most k, it is possible to decide in polynomial time whether G has a matching M such that $G - M$ is bipartite, and to find such a matching if it exists.*

Proof. Let G be as in the statement. A dominating set of order at most k can be found in time $O(n^k)$. Let D be such a dominating set of G of order at most k. Let $\mathscr{P}_{\mathscr{D}}$ be the set of all bipartitions P_D of D into sets A_D and B_D, such that $D[A_D]$ and $D[B_D]$ do not have any vertex of degree 2. Note that $|\mathscr{P}_{\mathscr{D}}| = O(2^k)$.

For each $P_D \in \mathscr{P}_{\mathscr{D}}$. We partition all of the other vertices $v \in V(G) \setminus V(D)$ in such a way: (i) If $d_{A_D}(v) \geq 2$ and $d_{B_D}(v) \geq 2$, then P_D is not a valid partition; (ii) If $d_{A_D}(v) \geq 2$, then $A_D \leftarrow A_D \cup \{v\}$; (iii) If $d_{B_D}(v) \geq 2$, then $B_D \leftarrow B_D \cup \{v\}$.

Iteratively applying these operations, we allocate the vertices in $V(G) \setminus V(D)$ as described above into the respective sets A_D e B_D, or we stop if it is not possible to acquire a valid bipartition. After that, $V(G) \setminus V(A_D \cup B_D)$ can be partitioned into three sets as follows: $X = \{u \in V(G) \setminus V(A_D \cup B_D) : d_{A_D}(u) = 1 \text{ and } d_{B_D}(u) = 0\}$; $Y = \{u \in V(G) \setminus V(A_D \cup B_D) : d_{A_D}(u) = 0 \text{ and } d_{B_D}(u) = 1\}$; $Z = \{u \in V(G) \setminus V(A_D \cup B_D) : d_{A_D}(u) = 1 \text{ and } d_{B_D}(u) = 1\}$;

Since every vertex in $V(G) \setminus V(D)$ has a neighbor in D, it follows that the neighborhood of all the vertices of $X \cup Y \cup Z$ in $A_D \cup B_D$ is in D. In this way, we can make a choice of a matching M_D to be removed, such that all of the vertices of $X \cup Y \cup Z$ in $A_D \cup B_D$ are allocated in $A_D \cup B_D$ and $G - M_D$ must be bipartite. Note that there are $(n-k)^k$ possibilities of choices for M_D. □

Corollary 2. ODD DECYCLING MATCHING *can be solved in polynomial time for P_5-free graphs.*

3.3 (Claw, Paw)-free Graphs

Lemma 5. *If $G \in \mathscr{BM}$ is a claw-free graph, then $\Delta(G) \leq 5$.*

Proof. Suppose that G has a vertex v of degree at least 6, such that $\{v_1, v_2, v_3, v_4, v_5, v_6\} \subseteq N_G(v)$. Since G is claw-free, then either $G[N_G(v)]$ has exactly two connected components, which must be cliques, or itself is connected. In the first case, since G is K_5-free, it follows that each connected component of $G[N_G(v)]$ has size at most 3. Moreover, by Lemma 1(ii), if both have size at least 3, then $G \notin \mathscr{BM}$, a contradiction.

Now suppose that $G[N_G(v)]$ is connected. If $G[N_G(v)]$ has a $P_5 = abcde$ as a subgraph, then it is not difficult to see that the only odd decycling matching M of $G[\{v\} \cup \{a,b,c,d,e\}]$ should have the edges ab, vc and de. Moreover $ae \notin E(G)$, otherwise $G[\{v\} \cup \{a,b,c,d,e\}]$ has a W_5 as a subgraph, a contradiction by Lemma 1(iii). In this way, any other vertex $f \in N(v)$ can be adjacent only to vertex c, since all of the vertices in $\{v,a,b,c,d,e\}$ are matched by M. Hence $G[\{v\} \cup \{a,e,f\}]$ induces a claw in G, a contradiction. On the other hand, necessarily $G[N_G(v)]$ has a P_3, otherwise there exist at least three connected components in $G[N_G(v)]$. By symmetry, suppose that v_1v_2 and $v_2v_3 \in E(G)$. By Lemma 1(ii) $G[\{v_4, v_5, v_6\}]$ has exactly one edge, say v_4v_5. Since $G[N_G(v)]$ is connected, the same has a path P of length at least 4 which connects v_6 to v_4 and v_5 by at least one vertex between v_1, v_2, and v_3. Since $G[N_G(v)]$ does not contain P_5, we can rename the vertices in $\{v_1, v_2, v_3, v_4, v_5, v_6\}$ such that $P = v_1v_2v_3v_4$. We know that $v_1v_4 \notin E(G)$, otherwise $P \cup \{v\}$ is a W_4, a contradiction by Lemma 1(iii). Hence v_5 and v_6 must be adjacent to at least one of v_1 and v_4, which creates a P_5 in $G[N_G(v)]$, a contradiction. □

By Lemma 5, it is not hard to enumerate all possible neighborhoods of a vertex v of a claw-free graph $G \in \mathscr{BM}$. In this way we can directly conclude the following lemma.

Lemma 6. *If $G \in \mathscr{BM}$ is a $(claw, paw)$-free graph then $\Delta(G) \leq 3$.*

Hence we can just apply Algorithm 1 to obtain a linear time algorithm for $(claw, paw)$-free graphs. Moreover, by Lemma 6 we can characterize the connected $(claw, paw)$-free graphs that admit an odd decycling matching.

Theorem 6. *If G is a connected $(claw, paw)$-free graph, then $G \in \mathscr{BM}$ if and only if G is isomorphic to a path, a cycle, a diamond or to a K_4.*

4 Fixed-Parameter Tractability

Now, we consider the parameterized complexity of ODD DECYCLING MATCHING.

Definition 1. *The clique-width of a graph G, denoted by $cwd(G)$, is defined as the minimum number of labels needed to construct G, using the following four operations [7]:*

1. *Create a single vertex v with an integer label ℓ (denoted by $\ell(v)$);*
2. *Disjoint union of two graphs (i.e. co-join) (denoted by $\oplus$);*
3. *Join by an edge every vertex labeled i to every vertex labeled j for $i \neq j$ (denoted by $\eta(i,j)$);*
4. *Relabeling all vertices with label i by label j (denoted by $\rho(i,j)$).*

Courcelle, Makowsky and Rotics [15] stated that for any graph G with clique-width bounded by a constant k, and for each graph property Π that can be formulated in a *monadic second order logic* ($MSOL_1$), there is a $f(cwd(G)).n$ algorithm that decides if G satisfies Π (cf. [12–16]). In this monadic second-order graph logic known as $MSOL_1$, the graph is described by a set of vertices V and a binary adjacency relation $edge(.,.)$, and the graph property in question may be defined in terms of sets of vertices of the given graph, but not in terms of sets of edges.

Theorem 7. ODD DECYCLING MATCHING *is in FPT when parameterized by the clique-width.*

Proof. Remind that the problem of determining whether G has an odd decycling matching is equivalent to determine whether G admits an $(1,1)$-coloring, which is a 2-coloring of $V(G)$ in which each color class induces a graph of maximum degree at most 1 (cf. [25]). Thus, using Courcelle, Makowsky and Rotics's meta-theorem based on monadic second order logic for graphs G with bounded clique-width [15], it is enough to observe that the property "G has an $(1,1)$-coloring" is $MSOL_1$-expressible. □

From Theorem 7 we can solve several interesting classes of graphs in polynomial time, as for example distance-hereditary graphs, series-parallel graphs, control flow graphs, and some subclasses of planar graphs such as outerplanar graphs, Halin graphs and Apollonian networks [4, 7, 8, 21, 30]. In addition, since clique-width generalizes several graph parameters [24], we have the following corollary.

Corollary 3. ODD DECYCLING MATCHING *is fixed-parameter tractable when parameterized by the following parameters: neighborhood diversity; treewidth; pathwidth; feedback vertex set; and vertex cover.*

MSOL's meta-theorem is a good classification tool, however it does not provide a precise running time bound. Next we present some exact upper bounds.

Theorem 8. ODD DECYCLING MATCHING *admits a $2^{O(vc(G))}.n$ algorithm, where $vc(G)$ is the vertex cover number.*

Theorem 9. ALLOWED ODD DECYCLING MATCHING *admits a kernel with at most $2.nd(G)$ vertices when parameterized by neighborhood diversity.*

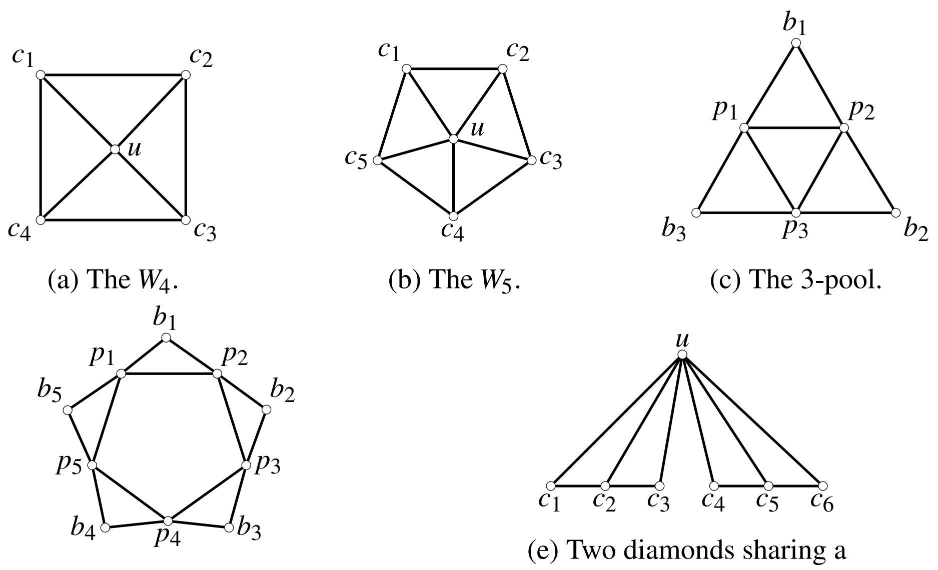
References

1. Abdullah, A.: On graph bipartization. In: Circuits and Systems, 1992. ISCAS '92. Proceedings., 1992 IEEE International Symposium on, vol. 4, pp. 1847–1850 (1992). DOI 10.1109/ISCAS.1992.230393
2. Alon, N., Stav, U.: Hardness of edge-modification problems. Theoretical Computer Science **410**(4749), 4920 – 4927 (2009). DOI http://dx.doi.org/10.1016/j.tcs.2009.07.002
3. Bazgan, C., Tuza, Z.: Combinatorial 5/6-approximation of max cut in graphs of maximum degree 3. Journal of Discrete Algorithms **6**(3), 510 – 519 (2008). DOI http://dx.doi.org/10.1016/j.jda.2007.02.002. URL http://www.sciencedirect.com/science/article/pii/S1570866707000822
4. Bodlaender, H.L.: A partial k-arboretum of graphs with bounded treewidth. Theoretical computer science **209**(1), 1–45 (1998)
5. Bondy, J.A., Locke, S.C.: Largest bipartite subgraphs in triangle-free graphs with maximum degree three. Journal of Graph Theory **10**(4), 477–504 (1986). DOI 10.1002/jgt.3190100407. URL http://dx.doi.org/10.1002/jgt.3190100407
6. Borodin, O., Kostochka, A., Yancey, M.: On 1-improper 2-coloring of sparse graphs. Discrete Mathematics **313**(22), 2638–2649 (2013). DOI http://dx.doi.org/10.1016/j.disc.2013.07.014
7. Brandstädt, A., Dragan, F.F., Le, H.O., Mosca, R.: New graph classes of bounded clique-width. Theory of Computing Systems **38**(5), 623–645 (2005)
8. Brandstädt, A., Spinrad, J.P., et al.: Graph classes: a survey, vol. 3. Siam (1999)
9. Burzyn, P., Bonomo, F., Durán, G.: Np-completeness results for edge modification problems. Discrete Applied Mathematics **154**(13), 1824 – 1844 (2006). DOI http://dx.doi.org/10.1016/j.dam.2006.03.031
10. Camby, E., Schaudt, O.: A new characterization of P_k-free graphs. Algorithmica **75**(1), 205–217 (2016). DOI 10.1007/s00453-015-9989-6. URL http://dx.doi.org/10.1007/s00453-015-9989-6
11. Choi, H.A., Nakajima, K., Rim, C.S.: Graph bipartization and via minimization. SIAM Journal on Discrete Mathematics **2**(1), 38–47 (1989). DOI 10.1137/0402004
12. Courcelle, B.: The monadic second-order logic of graphs. i. recognizable sets of finite graphs. Information and Computation **85**(1), 12 – 75 (1990). DOI http://dx.doi.org/10.1016/0890-5401(90)90043-H. URL http://www.sciencedirect.com/science/article/pii/089054019090043H
13. Courcelle, B.: The expression of graph properties and graph transformations in monadic second-order logic. Handbook of Graph Grammars **1**, 313–400 (1997)
14. Courcelle, B., Engelfriet, J.: Graph structure and monadic second-order logic, vol. 138. Citeseer (2011)
15. Courcelle, B., Makowsky, J.A., Rotics, U.: Linear time solvable optimization problems on graphs of bounded clique-width. Theory of Computing Systems **33**(2), 125–150 (2000). DOI 10.1007/s002249910009. URL http://dx.doi.org/10.1007/s002249910009
16. Courcelle, B., Mosbah, M.: Monadic second-order evaluations on tree-decomposable graphs. Theoretical Computer Science **109**(12), 49 – 82 (1993). DOI http://dx.doi.org/10.1016/0304-3975(93)90064-Z. URL http://www.sciencedirect.com/science/article/pii/030439759390064Z
17. Erdös, P.: On some extremal problems in graph theory. Israel Journal of Mathematics **3**(2), 113–116 (1965). DOI 10.1007/BF02760037. URL http://dx.doi.org/10.1007/BF02760037
18. Furmańczyk, H., Kubale, M., Radziszowski, S.: On bipartization of cubic graphs by removal of an independent set. Discrete Applied Mathematics (2015). DOI http://dx.doi.org/10.1016/j.dam.2015.10.036. In Press, Corrected Proof

19. Garey, M., Johnson, D., Stockmeyer, L.: Some simplified np-complete graph problems. Theoretical Computer Science **1**(3), 237 – 267 (1976). DOI http://dx.doi.org/10.1016/0304-3975(76)90059-1
20. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., New York, NY, USA (1979)
21. Golumbic, M.C., Rotics, U.: On the clique-width of some perfect graph classes. International Journal of Foundations of Computer Science **11**(03), 423–443 (2000)
22. Guillemot, S., Havet, F., Paul, C., Perez, A.: On the (non-)existence of polynomial kernels for p-free edge modification problems. Algorithmica **65**(4), 900–926 (2012). DOI 10.1007/s00453-012-9619-5
23. Hopcroft, J., Tarjan, R.: Efficient planarity testing. J. ACM **21**(4), 549–568 (1974). DOI 10.1145/321850.321852
24. Lampis, M.: Algorithmic meta-theorems for restrictions of treewidth. Algorithmica **64**(1), 19–37 (2012)
25. Lovász, L.: On decomposition of graphs. Studia Scientiarum Mathematicarum Hungarica **1**, 237–238 (1966)
26. Moret, B.M.E.: Planar nae3sat is in p. SIGACT News **19**(2), 51–54 (1988). DOI 10.1145/49097.49099. URL http://doi.acm.org/10.1145/49097.49099
27. Mulzer, W., Rote, G.: Minimum-weight triangulation is np-hard. Journal of the ACM (JACM) **55**(2), 11 (2008)
28. Natanzon, A., Shamir, R., Sharan, R.: Complexity classification of some edge modification problems. Discrete Applied Mathematics **113**(1), 109 – 128 (2001). DOI http://dx.doi.org/10.1016/S0166-218X(00)00391-7. Selected Papers: 12th Workshop on Graph-Theoretic Concepts in Computer Science
29. Schaefer, T.J.: The complexity of satisfiability problems. In: Proceedings of the Tenth Annual ACM Symposium on Theory of Computing, STOC '78, pp. 216–226. ACM, New York, NY, USA (1978)
30. Thorup, M.: All structured programs have small tree width and good register allocation. Information and Computation **142**(2), 159–181 (1998)
31. Yannakakis, M.: Node-and edge-deletion np-complete problems. In: Proceedings of the Tenth Annual ACM Symposium on Theory of Computing, STOC '78, pp. 253–264. ACM, New York, NY, USA (1978). DOI 10.1145/800133.804355
32. Yannakakis, M.: Edge-deletion problems. SIAM Journal on Computing **10**(2), 297–309 (1981)
33. Yarahmadi, Z., Ashrafi, A.R.: A fast algorithm for computing bipartite edge frustration number of $(3,6)$-fullerenes. Journal of Theoretical and Computational Chemistry **13**(02), 1450,014 (2014). DOI 10.1142/S021963361450014X

APPENDIX

Fig. 6 shows some others examples of forbidden subgraphs.



(a) The W_4. (b) The W_5. (c) The 3-pool.

(d) The 5-pool. (e) Two diamonds sharing a vertex of degree 3.

Fig. 6: Some examples of forbidden subgraphs.

Proof of Lemma 1.

Proof. (i) Let $G \in \mathscr{BM}$ be a graph that contains a diamond D as a subgraph, such that $V(D) = \{u, v_1, v_2, v_3\}$ and $d_D(u) = d_D(v_2) = 3$. We can see that $M \cap E(D)$ equals to exactly one of the following sets: $\{uv_1, v_2v_3\}$, $\{v_1v_2, uv_3\}$, $\{uv_2\}$. For each of such sets, both u and v_2 are matched by M. Hence M cannot contain any edge $e \notin E(G[V(D)])$ incident to only u or v_2.

(ii) Let $v \in V(G)$ such that $G[N_G(v)]$ contains two disjoint P_3, P and P'. It follows that $G[\{v\} \cup P]$ and $G[\{v\} \cup P']$ are diamonds that share a vertex of degree at least three. By (i) the statement holds.

(iii) Suppose for a contradiction that G contains a subgraph H isomorphic to a wheel graph W_k, $k \geq 4$. Let $V(H) = \{u, v_1, v_2, \ldots, v_{k-1}, v_k\}$, such that u is adjacent to all vertices of the cycle $C = v_1v_2 \ldots v_kv_1$. If $k \geq 6$, then u contains two disjoint P_3 in its neighborhood, and thus it follows by (ii) that $k \leq 5$. In this case, it can be easily verified that W_4 and W_5 are forbidden subgraphs.

(iv) Suppose, for a contradiction, that G contains a subgraph H isomorphic to a k-pool, for some odd $k \geq 3$. Let $C = \{p_1p_2 \ldots p_kp_1\}$ be its internal cycle and let $B = \{b_1, b_2, \ldots, b_k\}$ be the vertices of the border of H, such that $\{p_ib_i, p_{i+1}b_i\} \subset E(H)$, for all $1 \leq i \leq k$ modulo k. Clearly M must contain some edge of C and one edge of every triangle $p_i b_i p_{i+1}$. W.l.o.g., consider $p_1 p_2 \in M \cap E(C)$. This implies that M contains no edge in $\{p_2 b_2, p_2 p_3, p_1 b_k, p_1 p_k\}$. Therefore, $p_k b_k$ and $p_3 b_2$ must be in M, which forbids two more edges from the triangles $p_{k-1} b_{k-1} p_k$ and $p_3 b_3 p_4$. Continuing this process, it follows that $c_{\lfloor \frac{k+3}{2} \rfloor}$, which is at the same distance of p_1 and p_2 in C, must contain two incident edges in M, a contradiction. □

Proof of Theorem 1.

Proof. Let A be a maximal independent set of G. Let $B = V(G) \setminus A$. In this case, every vertex of A is of type $(k,0)$ and there is no vertex in B of type $(0,k)$, $k \in \{1,2,3\}$. Therefore, if there exists a vertex v of type (a,b) with $a < b$, then it must be in B and be of type $(1,2)$. In order to prove the correctness of Algorithm 1, it is sufficient to show that the operations on lines 7–8 and 10–11 do not generate vertices of type (a,b) with $a < b$.

Let $\{u\} = N_{G[A]}(v)$. If u is of type $(3,0)$, then v is moved from B to A by lines 7–8. In this case, it follows that both u and v are vertices of type $(2,1)$ after the line 8. If u is not of type $(3,0)$, then the lines 10–11 modify the types of u and v as follows.

– If u is of type $(1,1)$, then u and v are modified to type $(2,0)$ and $(3,0)$, respectively;
– If u is of type $(1,0)$, then u and v are modified to type $(1,0)$ and $(3,0)$, respectively;
– If u is of type $(2,0)$, then u and v are modified to type $(1,1)$ and $(3,0)$, respectively;
– If u is of type $(2,1)$, then u and v are modified to type $(2,1)$ and $(3,0)$, respectively.

We can see that each neighbor w of u in the same part $X \in \{A,B\}$ of u loses exactly one neighbor (that is u) in $G[X]$. Moreover, w receives at most one new neighbor (that is v) in $G[X]$. The same occurs for every neighbor of v in $V(G) \setminus X$. Therefore, in any case it is not obtained vertices of type (a,b) with $a < b$, which implies that the Algorithm 1 finishes. □

Proof of Theorem 2.

Proof. Since verifying whether a graph is planar can be done in linear time [23], as well as whether a formula in 3-CNF has a truth assignment, both problems are in NP.

Let F be a Boolean formula in 3-CNF such that $X = \{x_1, x_2, \ldots, x_n\}$ denotes the set of variables and $C = \{c_1, c_2, \ldots, c_m\}$ is the set of clauses of F. We construct a formula F' from F as follows. For a vertex $x_i \in V(G_F[X])$, let $d_{G_F}(x_i)$ be the degree of x_i in G_F. For such a variable x_i with $d_{G_F}(x_i) = k \geq 3$, we create k new clauses c_i^j of size 2, and k new variables x_i^z as follows:

$$c_i^j = \begin{cases} \left(x_i^j, \overline{x_i^{j+1}}\right) & \text{, if } j \in \{1, \ldots, k-1\}; \\ \left(x_i^k, \overline{x_i^1}\right) & \text{, if } j = k. \end{cases}$$

In addition, we replace the j^{th} $(1 \leq j \leq k)$ occurrence of the variable $x_i \in X$ by an occurrence of a variable x_i^j, where a literal x_i (resp. $\overline{x_i}$) is replaced by a literal x_i^j $\left(\text{resp. } \overline{x_i^j}\right)$.

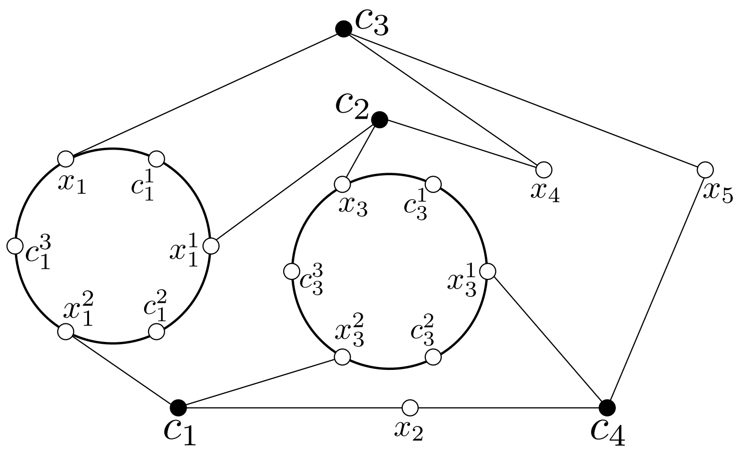


Fig. 7: The associated graph $G_{F'}$ obtained from $F = (x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_3 \vee x_4) \wedge (x_1 \vee x_4 \vee x_5) \wedge (x_2 \vee x_3 \vee x_5)$. The black vertices correspond to clauses.

Let S be the set of all vertices $x_i \in V(G_F[X])$ with $d_{G_F}(x_i) = k \geq 3$. For such a vertex $x_i \in S$, let $X_i = \{x_i^1, \ldots, x_i^k\}$ and $C_i = \{c_i^1, \ldots, c_i^k\}$.

Note that, the associated graph $G_{F'}$ can be obtained from G_F by replacing the corresponding vertex of $x_i \in S$ by a cycle of length $2d_{G_F}(x_i)$ induced by the corresponding vertices of the new clauses in C_i and the new variables in X_i. In addition, for each $x_i \in S$ and $c_j \in N_{G_F}(x_i)$ an edge $x_i^t c_j$ is added in $E(G_{F'})$, such that every corresponding vertex $x_i^t \in X_i$ has exactly one neighbor $c_j \notin C_i$. Fig. 7 shows an example of the transformation for a Boolean formula.

As we can see, every variable x occurs at most 3 times in the clauses of F', since every variable x_i with $d_{G_F}(x_i) \geq 3$ is replaced by $d_{G_F}(x_i)$ new variables that are in exactly 3 clauses of F'. By the construction, each literal occurs at most twice. Moreover, if F has no negative literals, then only the new variables have a negated literal and each one occurs exactly once in F'.

Now, it remains to show that if G_F is planar then we can construct F' as a planar formula. Consider a planar embedding Ψ of G_F, we construct $G_{F'}$ replacing each corresponding vertex $x_i \in S$ by a cycle of length $2d_{G_F}(x_i)$, as described above. After that, in order to preserve the planarity, we can follow the planar embedding Ψ to add a matching between vertices corresponding to variables in such a cycle and vertices corresponding to clauses $c_j \notin C_i$ and that $x_i \in c_j$. Such a matching indicates in which clause of C_i a given new variable will replace x_i in F'. Thus, without loss of generality, if G_F is planar then we can assume that F' is planar as well.

Let F be an instance of NAE-3SAT (resp. POSITIVE PLANAR 1-IN-3-SAT) such that $X = \{x_1, \ldots, x_n\}$ denotes its set of variables and $C = \{c_1, c_2, \ldots, c_m\}$ its set of clauses. Let F' be the formula obtained from F by the above construction. As we can observe, for any truth assignment of F', all $x_i^t \in X_i$ (for a given variable x_i of F) have the same value. Therefore, any clause of F' containing exactly two literals has true and false values. At this point, it is easy to see that F has a not-all-equal (resp. 1-in-3) truth assignment if and only if F' has a not-all-equal (resp. 1-in-3) truth assignment. □

Proof of Lemma 3.

Proof. Let M be an odd decycling matching of G. Suppose for a contradiction that there exists an edge e incident to v, such that e contains an endvertex not in H. In this case, we get that vh_1 and vh_4 does not belong to M, which implies that $h_1h_4 \in M$. By the triangle $h_1h_2h_5$, it follows that h_2h_5 must be in M. Hence the cycle $vh_1h_2h_3h_4v$ remains in $G-M$, a contradiction.

Now suppose that $vh_4 \in M$. In this case, the edge h_1h_2 cannot be in M, otherwise the cycle $h_1h_4h_3h_2h_5h_1$ survives in $G-M$. In the same way, the edge $h_1h_5 \notin M$, otherwise the cycle $h_1h_2h_5h_3h_4h_1$ is not destroyed by M. Therefore we get that h_2h_5 must be in M, which implies that $h_3h_6 \in M$. Hence the cycle $h_5h_3h_4h_7h_6h_5$ belongs to $G-M$. Since the triangle $h_1h_2h_5$ has no edge in M, it is not destroyed by M, a contradiction.

Finally, we get that vh_1 must be in M, which implies that $h_2h_5 \in M$ as well. Therefore, it follows that h_3h_6 must be in M. Hence h_4h_7 also must be in M, which turns the graph bipartite. Since all choices of the edges of M are necessary, we get that there is only one possible odd decycling matching of H, which is perfect. Fig. 1b shows such a matching. This concludes the proof. □

Proof of Lemma 4.

Proof. Let $C = p_1p_2\dots p_kp_1$ be the internal cycle of G and let b_i be the i-th-border of G, such that $N_G(b_i) = \{p_i, p_{i+1}\}$, $1 \le i \le k-1$. Since C has odd length, it follows that every odd decycling matching of G contains at least one edge of C.

Suppose for a contradiction that G has an odd decycling matching M containing p_1p_k. In this case, we get that the edges in $\{p_1p_2, p_1b_1, p_kp_{k-1}, p_kb_{k-1}\}$ cannot be in M. Therefore M must contain the edges b_1p_2 and $b_{k-1}p_{k-1}$. In the same way, we can see that the edges $\{p_2p_3, p_2b_2, p_{k-1}p_{k-2}, p_{k-1}b_{k-2}\}$ are not in M. Hence, it can be seen that all edges indent to $p_{\frac{k-1}{2}}$ are forbidden to be in M, which implies that the triangles $p_{\frac{k-2}{2}}p_{\frac{k-1}{2}}b_{\frac{k-2}{2}}$ and $p_{\frac{k-1}{2}}p_{\frac{k+1}{2}}b_{\frac{k-1}{2}}$ have no edge in M, a contradiction by the choice of M.

Let p_ip_{i+1} be an edge of C contained in an odd decycling matching M of G. In a same fashion, the edges in $\{p_ip_{i-1}, p_ib_{i-1}, p_ip_{i+1}, p_ib_{i+1}\}$ cannot be in M. Following this pattern, we can see that every edge p_jb_j must be in M, for every $1 \le j \le i-1$. Furthermore, it follows that $b_zp_{z+1} \in M$, for every $i+1 \le z \le k-1$. Since M contains one edge of every triangle of G, it follows that M is unique, for every edge p_ip_{i+1}. Finally, such an odd decycling matching contains only one edge of C. □

Proof of Corollary 1.

Proof. Let G be the graph obtained by the construction in Theorem 3, next we show how to obtain a planar graph of maximum degree 4. Since the only vertices of degree 5 of G are those p_i^4 in the variable gadgets, we present a slightly modified variable gadget, which is depicted in Fig 8. In fact, we just modify the head graph. In Fig. 1 we can see that the vertex h_6 has degree 3, which allows us to use it to connect the variable gadget to the clause one. Fig. 9 shows the possible odd decycling matching of the modified variable gadget. Since such configurations are analogous to those of the original variable gadget, with respect to the vertex that connect to clause gadgets, we obtain our main result of this section. □

Proof of Theorem 6.

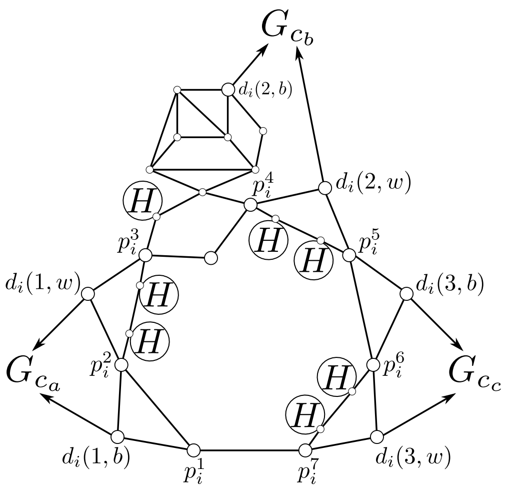


Fig. 8: The modified variable gadget.

Proof. Let G be a connected $(claw, paw)$-free graph. Clearly, if G is isomorphic to a path, a cycle, a diamond or to a K_4, then $G \in \mathscr{BM}$.

Now consider $G \in \mathscr{BM}$. By Lemma 6 we know that does not exist any vertex of degree 4 in G.

If all of the vertices of G have degree 2, then G is isomorphic to a cycle.

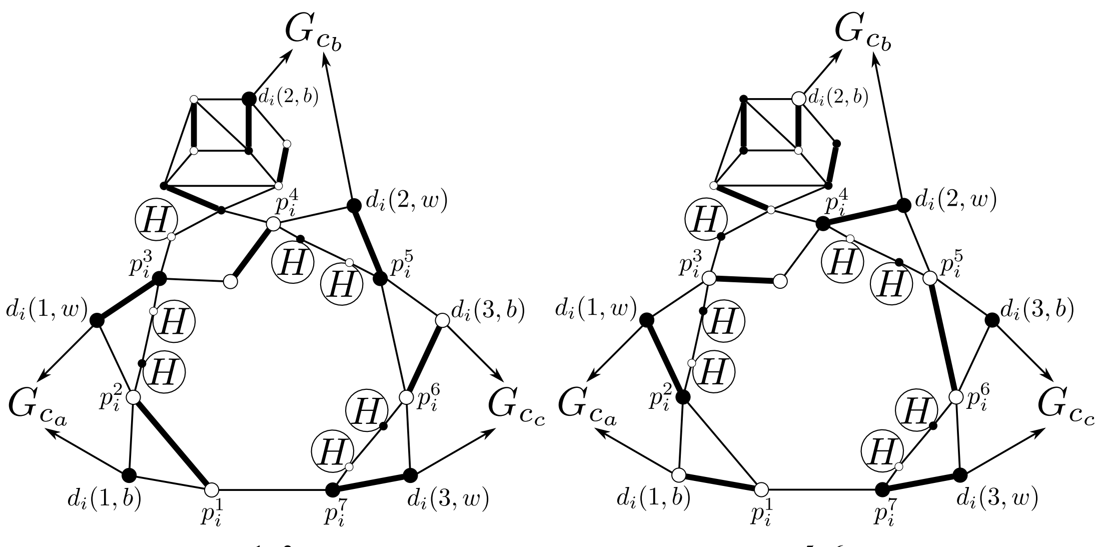
If G is the trivial graph, then the theorem follows. Let v be a vertex of degree 1 in G. It follows that either G is a path of length at least one, or there is a vertex u of degree three. Consider u such that it is such a vertex closest to v in G. Let $w \in N_G(u)$ be in the path P from v to u and let u_1 and u_2 be its two neighbors except for w in G. Since $d_G(w) \leq 2$, we get that w is not adjacent neither to u_1, nor to u_2. Thus $u_1u_2 \in E(G)$, since G is claw-free. In this way $G[wuu_1u_2]$ is isomorphic to a paw, a contradiction. Hence G must be a path.

Finally suppose that G has a vertex v of degree 3 and that G is not isomorphic to a K_4. In this way it follows that $G[\{v\} \cup N_G(v)]$ is isomorphic to a diamond. Let $N_G(v) = \{v_1, v_2, v_3\}$, such that $d_G(v_2) = 3$. Since v and v_2 have degree 3, they cannot be adjacent to any other vertex. Suppose that v_1 has a neighbor $u \notin \{v, v_2\}$. Since uv, $uv_2 \notin E(G)$, then $G[\{u, v, v_1, v_2\}]$ is isomorphic to a paw, a contradiction. It follows in a similar way that $N_G(v_3) = \{v, v_2\}$. Hence G is isomorphic to a diamond, which concludes the proof. $\square$

A monadic second order logic formula $\varphi(G)$ such that $G \in \mathscr{BM} \Leftrightarrow \varphi(G)$:

$$\begin{aligned}\exists\, S_1, S_2 \subseteq V(G)\ :\ & (S_1 \cap S_2 = \emptyset) \wedge \\ & (S_1 \cup S_2 = V(G)) \wedge \\ & (\forall\, v_1 \in S_1[\,\nexists\, u_1, w_1 \in S_1 :\ (u_1 \neq w_1) \wedge edge(u_1, v_1) \wedge edge(w_1, v_1)]) \wedge \\ & (\forall\, v_2 \in S_2[\,\nexists\, u_2, w_2 \in S_2 :\ (u_2 \neq w_2) \wedge edge(u_2, v_2) \wedge edge(w_2, v_2)])\end{aligned}$$

Proof of Theorem 8.



(a) $p_i^1 p_i^2 \in M$ (b) $p_i^5 p_i^6 \in M$

Fig. 9: All possible configurations given by the removal of an odd decycling matching M from the modified variable gadget.

Proof. Let S be a vertex cover of G such that $|S| = vc(G)$. The algorithm follows in a similar way to Algorithm 5. Let $\mathscr{P}_{\mathscr{S}}$ be the set of all bipartitions P_S of S into sets A_S and B_S, such that $S[A_S]$ and $S[B_S]$ do not have any vertex of degree 2. For each $P_S \in \mathscr{P}_{\mathscr{S}}$, we will check if an odd decycling matching of G can be obtained from P_S by applying the following operations: For each vertex $v \in V(G) \setminus V(S)$ do (i) If $d_{A_S}(v) \geq 2$ and $d_{B_S}(v) \geq 2$, then P_S is not a valid partial partition; (ii) If $d_{A_S}(v) \geq 2$, then $A_S \leftarrow A_S \cup \{v\}$; (iii) If $d_{B_S}(v) \geq 2$, then $B_S \leftarrow B_S \cup \{v\}$. After that, if for all vertices the first condition is not true, then $V(G) \setminus V(A_S \cup B_S)$ can be partitioned into three sets: $X = \{u \in V(G) \setminus V(A_S \cup B_S) : d_{A_S}(u) = 1 \text{ and } d_{B_S}(u) = 0\}$; $Y = \{u \in V(G) \setminus V(A_S \cup B_S) : d_{A_S}(u) = 0 \text{ and } d_{B_S}(u) = 1\}$; $Z = \{u \in V(G) \setminus V(A_S \cup B_S) : d_{A_S}(u) = 1 \text{ and } d_{B_S}(u) = 1\}$;

Since $V(G) \setminus V(S)$ is an independent set, it follows that all edges of vertices in $X \cup Y$ can remain in the graph G. For each $z \in Z$, denote by $a_z \in A_S$ and $b_z \in B_S$ the neighbors of z in G. Now, we apply a bounded search tree algorithm. While $G[A_S]$ and $G[B_S]$ have both maximum degree equal to one, and $Z \neq \emptyset$ do. Remove a vertex $z \in Z$ and apply recursively the algorithm for the following cases: z is added to A_S, and all vertices in $Z \cap N(a_z)$ is added to B_S; z is added to B_S, and all vertices in $Z \cap N(b_z)$ is added to A_S.

T has height at most $vc(G) + 1$. Finally, if T has a leaf representing a configuration with $G[A_S]$ and $G[B_S]$ having both maximum degree equal to one, and $Z = \emptyset$ then G has an odd decycling matching. □

Proof of Theorem 9.

Proof. Given an instance (G,F) of ALLOWED ODD DECYCLING MATCHING such that G is a graph and $F \subseteq E(G)$ a set of forbidden edges. The kernelization algorithm consists on applying the following reduction rules:

1. *If G contains a K_5, then G has no allowed odd decycling matching; otherwise*

2. *If a part V_i induces a K_3 and exist two vertices in $V(G) \setminus V_i$ adjacent to V_i, then G has no allowed odd decycling matching; otherwise*
3. *If a subgraph of G induces either a K_3 or a K_4 and does not admit an allowed odd decycling matching, then G has no allowed odd decycling matching; otherwise*
4. *Remove all parts isomorphic to a K_4;*
5. *Remove all isolated parts isomorphic to a K_3;*
6. *If V_i is a part that induces a K_3 and $v \in V(G) \setminus V_i$ is adjacent to V_i (note that $\{v\}$ is a part), then remove V_i and $F \leftarrow F \cup \{uv : u \in N_G(v) \setminus V_i\}$;*
7. *If a part V_i induces an independent set of size at least 3, then contract it into a single vertex v_i (without parallel edges) and forbids all of its incident edges;*

It is easy to see that all reduction rules can be applied in polynomial time, and after applying them any remaining part has size at most two. As the resulting graph G' has $nd(G') \leq nd(G)$ then $|V(G')| \leq 2.nd(G)$. Thus, it remains to prove that the application of each reduction rule is correct. As K_5 and $K_5 - e$ are forbidden subgraphs, and any odd decycling matching of a K_4 is a perfect matching then rules 1,2,3,4,5 and 6 can be applied in such an order. Finally, the correctness of rule 7 follows from the following facts: (i) if G' has an allowed odd decycling matching, then G has also an allowed odd decycling matching, because bipartite graph class is closed under the operation of replacing vertices by a set of false twins, which have the same neighborhood as the replaced vertex; (ii) if G' does not admit an allowed odd decycling matching then G also does not admit an allowed odd decycling matching, because if a contracted single vertex v_i is in an odd cycle in G', then even replacing v_i by V_i ($|V_i| \geq 3$) and removing some incident edges of V_i which form a matching, some vertex of V_i remains to an odd cycle. □