UNIVERSIDADE FEDERAL FLUMINENSE

Alan Diêgo Aurélio Carneiro

On the Knot-Free Vertex Deletion Problem: A Parameterized Complexity Analysis

NITERÓI

2020

UNIVERSIDADE FEDERAL FLUMINENSE

Alan Diêgo Aurélio Carneiro

On the Knot-Free Vertex Deletion Problem: A Parameterized Complexity Analysis

Tese de Doutorado apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do Grau de Doutor em Computação. Área de concentração: Algoritmos e Otimização

Orientadores:
Fábio Protti
Uéverton dos Santos Souza

NITERÓI

2020

Ficha catalográfica automática - SDC/BEE
Gerada com informações fornecidas pelo autor

C289o Carneiro, Alan Diêgo Aurélio
On the Knot-Free Vertex Deletion Problem : A Parameterized Complexity Analysis / Alan Diêgo Aurélio Carneiro ; Fábio Protti, orientador ; Uéverton dos Santos Souza, coorientador. Niterói, 2020.
88 f. : il.

Tese (doutorado)-Universidade Federal Fluminense, Niterói, 2020.

DOI: http://dx.doi.org/10.22409/PGC.2020.d.03493711131

1. Knot. 2. FPT. 3. W[1]-difícil. 4. Treewidth. 5. Produção intelectual. I. Protti, Fábio, orientador. II. Souza, Uéverton dos Santos, coorientador. III. Universidade Federal Fluminense. Instituto de Computação. IV. Título.

CDD -

Bibliotecária responsável: Fabiana Menezes Santos da Silva - CRB7/5274

Alan Diêgo Aurélio Carneiro

On the Knot-Free Vertex Deletion Problem: A Parameterized Complexity Analysis

Tese de Doutorado apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do Grau de Doutor em Computação. Área de concentração: Algoritmos e Otimização

Aprovada em 24 de março de 2020.

BANCA EXAMINADORA

Prof. Fábio Protti - Orientador, UFF

Uéverton dos Santos Souza

Prof. Uéverton dos Santos Souza - Orientador, UFF

Luis Antonio B Kowada

Prof. Luis Antonio Brasil Kowada - UFF

Prof. Luís Felipe Ignácio Cunha - UFF

Prof. Jayme Luiz Szwarcfiter - UFRJ

Rudini M. Sampaio

Prof. Rudini Menezes Sampaio - UFC

Niterói

2020

For Nicole, Arthur and Alana.

Resumo

Um knot em um grafo direcionado G é um subgrafo fortemente conexo Q de G com pelo menos dois vértices, tal que, nenhum vértice em $V(Q)$ possui aresta direcionada a um vértice em $V(G)\setminus V(Q)$. O Knot é uma estrutura importante já que caracteriza a existência de deadlocks em um modelo de computação distribuída clássico, chamado modelo Ou. A detecção de deadlocks está intimamente relacionada ao reconhecimento de grafos livres de knots da mesma maneira que a resolução de deadlock está intimamente relacionada ao problema de Eliminação de Knots por Deleção de Vértices (Knot-Free Vertex Deletion (KFVD)), que consiste em determinar se dado um grafo G e um inteiro k, G possui um subconjunto de vértices $S \subseteq V(G)$ e $|S| \leq k$ tal que $G[V \setminus S]$ não contém knot.

Nesta tese, primeiramente foi realizada uma revisão da literatura sobre a complexidade computacional do problema de resolução de deadlock nos modelos clássicos de computação distribuída. Foram apresentados que: o problema de Eliminação de Knots por Deleção de Arcos pode ser resolvido em tempo $O(n)$; KFVD é NP-difícil mesmo quando o grafo de entrada é fortemente conexo ou bipartido, planar e com grau máximo 4; KFVD pode ser resolvido em tempo $O(m\sqrt{n})$ quando o grafo de entrada é sub-cúbico. Além disso, é apresentada uma equivalência entre as versões do problema de Eliminação de Knots por Deleção de Arcos/Vértices para grafos com pesos e grafos sem pesos.

Em seguida, é apresentada uma análise de complexidade parametrizada de granularidade fina para KFVD. Provamos que: KFVD é W[1]-difícil quando parametrizado pelo tamanho da solução k; pode ser solucionado em tempo $2^{k \log \varphi} n^{O(1)}$, mas assumindo a hipótese de tempo exponencial forte *(Strong Exponential Time Hypothesis (SETH))* não pode ser solucionado em tempo $(2-\epsilon)^{k \log \varphi} n^{O(1)}$, onde φ é o tamanho da maior componente fortemente conexa de G; pode ser resolvido em tempo $2^{\phi} n^{O(1)}$, mas assumindo a hipótese de tempo exponencial *(Exponential Time Hypothesis (ETH))* não pode ser resolvido em tempo $2^{o(\phi)} n^{O(1)}$, onde ϕ é o número de vértices com grau de saída no máximo k; a menos que $NP \subseteq coNP/poly$, KFVD não admite núcleo polinomial mesmo quando $\varphi = 2$ e parametrizado pelo tamanho da solução k.

Finalmente, considera-se parâmetros de largura onde provamos que: KFVD quando parametrizado pelo tamanho da solução k é W[1]-difícil mesmo quando o tamanho do maior caminho direcionado p, juntamente com o Kenny-width do grafo são limitados por constantes; é solucionável em tempo FPT quando parametrizado por clique-width; pode ser resolvido em tempo $2^{O(tw \log tw)} \times n$, mas, assumindo ETH não pode ser resolvido em tempo $2^{o(tw)} \times n^{O(1)}$, onde tw é a treewidth do grafo subjacente. Além disso, dado que o tamanho do conjunto mínimo de vértices de retroalimentação (directed feedback vertex set) dfv é um limite superior para o tamanho de um certificado de solução para KFVD, investigamos parametrizações por dfv, onde mostramos que KFVD pode ser solucionado em tempo FPT quando parametrizado tanto por $dfv+\kappa$ ou $dfv+p$, e adimite um algoritmo FPT quando parametrizado pela distância a um DAG tendo uma cobertura por caminhos limitada (outro parâmetro superior ao dfv).

Palavras-chave: Knot, FPT, W[1]-difícil, ETH, Treewidth, Parâmetros de largura.

Abstract

A knot in a directed graph G is a strongly connected subgraph Q of G with at least two vertices, such that no vertex in $V(Q)$ is an in-neighbor of a vertex in $V(G) \setminus V(Q)$. Knots are important graph structures because they characterize the existence of deadlocks in a classical distributed computation model, the so-called OR-model. Deadlock detection is correlated with the recognition of knot-free graphs, as well as deadlock resolution, is closely related to the Knot-Free Vertex Deletion (KFVD) problem, which consists of determining whether given a graph G and an integer k, G has a subset $S \subseteq V(G)$ and $|S| \leq k$ such that $G[V \setminus S]$ contains no knot.

In this thesis, first it is done a literature review reggarding the computational complexity of the deadlock resolution problem in the classical distributed computational models. It is shown that: the problem of Knot-Free Arc Deletion can be solved in $O(n)$ time; the KFVD is NP-hard even when the input graph is strongly connected or bipartite, planar and with maximum degree 4; KFVD can be solved in $O(m\sqrt{n})$ time when the input graph is sub-cubic. Also, an equivalence between the versions of the Knot-Free Arc/Vertex Deletion problems for weighted and for unweighted graphs is also presented.

Next, a fine-grained parameterized complexity analysis of KFVD is presented. It is shown that: KFVD is W[1]-hard when parameterized by the size of the solution k; it can be solved in $2^{k \log \varphi} n^{O(1)}$ time, but assuming Strong Exponential Time Hypothesis (SETH) it cannot be solved in $(2-\epsilon)^{k \log \varphi} n^{O(1)}$ time, where φ is the size of the largest strongly connected subgraph of G; and it can be solved in $2^{\phi} n^{O(1)}$ time, but assuming Exponential Time Hypothesis (ETH) it cannot be solved in $2^{o(\phi)} n^{O(1)}$ time, where ϕ is the number of vertices with out-degree at most k; unless $NP \subseteq coNP/poly$, KFVD does not admit polynomial kernel even when $\varphi = 2$ and k is the parameter.

Finally, we focus on width parameterizations where we show that: KFVD parameterized by the size of the solution k is W[1]-hard even when p, the length of a longest directed path of the input graph, as well as κ, its Kenny-width, are bounded by constants, and we remark that KFVD is para-NP-hard even considering many directed width measures as parameters, but in FPT when parameterized by clique-width; KFVD can be solved in $2^{O(tw \log tw)} \times n$ time, but assuming ETH it cannot be solved in $2^{o(tw)} \times n^{O(1)}$, where tw is the treewidth of the underlying undirected graph. Since the size of a minimum directed feedback vertex set (dfv) is an upper bound for the size of a minimum knot-free vertex deletion set, we investigate parameterization by dfv, and we show that KFVD can be solved in FPT-time parameterized by either $dfv+\kappa$ or $dfv+p$, and it admits an FPT-time algorithm by the distance to a DAG having bounded path cover (another parameter larger than dfv).

Keywords: Knot, FPT, W[1]-hard, ETH, Treewidth, Width parametrization.

List of Figures

List of Tables

Contents

Chapter 1

Introduction

A set of processes is in deadlock if each process of this set is blocked, waiting for a resource to be freed, which is controlled by another process this same set; that is, the processes cannot continue their execution, waiting for an event or a signal that only another process of this same set can send. In other words, a deadlock situation is characterized by the permanent impediment for a set of processes to proceed with their tasks due to a condition that blocks at least one essential resource to be acquired [8].

Deadlock is a common phenomenon when some kind of resource sharing is needed, such as: operating systems [87]; traffic intersections [30]; railway lines [80, 79]; and multi-robot cooperation [44], to name just a few examples. Although this problem is extensively studied in shared memory systems [30, 42], the problem remains difficult to solve and has several open questions.

The existence of deadlock in a graph representation of a system can be accounted by a specific graph structure which varies according to the model of the computation. For instance, in a classical deadlock model "AND", the existence of *cycles* characterize a deadlock in the input graph. For the "OR" model, the existence of a structure called *knot* accounts for the existence of deadlock in the input graph. This work we mainly regard the Knot-Free Vertex Deletion Problem, that is, a vertex deletion graph problem related to the deadlock resolution problem in the "OR" model.

1.1 Knot-Free Vertex Deletion Problem

A knot is an important graph structure with direct application in distributed computation. According to Barbosa [7], for $v_i \in V$ let T_i be the set of vertices that can be reached from

v_i through a directed path in G. A set of vertices $K \subseteq V$ is a *knot* in G if and only if S has at least two vertices and, for all $v_i \in S$, $T_i = S$. Another definition, by Misra and Chandy [74], a vertex v of a directed graph G is in a *knot* if for every vertex v_j reachable from v_i, v_i is reachable from v_j. Notice that, by definition, no member of a knot has a sink in its reachability set.

All strongly connected components on a graph G can be found through a topological ordering of G, since a knot is a strongly connected component C of cardinality $|C| > 1$ where there are no paths from a C vertex to any $G[V \setminus C]$ vertex, then, all the knots of a digraph can be identified in linear time as follows: first, find all the SCCs in linear time by running a depth-first search (Cormen et al. [32], pages 615-621); next, contract each SCC into a single vertex, obtaining an acyclic digraph H whose new sinks represent the knots of G.

the concept of Knot is useful in distributed computation, with application in deadlock detection and deadlock resolution, because they characterize the existence of deadlocks in a classical distributed computation model, the so-called OR-model [10]. Deadlock detection is correlated with the recognition of knot-free graphs as well as deadlock resolution is closely related to the Knot-Free Vertex Deletion (KFVD) problem, which consists of determining whether an input graph G has a subset $S \subseteq V(G)$ of size at most k such that $G[V \setminus S]$ contains no knot.

Formally the KFVD problem can be defined as follows.

Knot-Free Vertex Deletion (KFVD)
Instance: A directed graph $G = (V, E)$; and a positive integer k.
Question: Determine if G has a set $S \subseteq V(G)$ such that $\|S\| \leq k$ and $G[V \setminus S]$ is knot-free.

The focus of this work is to study the KFVD graph problem. It is interesting to point that KFVD is closely related to the Directed Feedback Vertex Set (DFVS) problem because of their relation with deadlocks, besides that, there are some structural similarities between them. The goal of DFVS is to obtain a directed acyclic graph (DAG) through vertex deletion (in such graphs *all* maximal directed paths end in a sink); the goal of KFVD is to obtain a knot-free graph, and in such graphs, for every vertex v, there is *at least one* maximal path containing v that ends in a sink. Finally, every directed feedback vertex set is a knot-free vertex deletion set; thus, the size of a minimum directed feedback vertex set is an upper bound for KFVD.

1.2 Objectives and Contribution

This work provides an analysis of the computational complexity of the Knot-Free Vertex Deletion (KFVD) problem. Given a directed graph G, we investigate vertex deletion problem whose goal is to obtain the minimum number of these removals in order to turn G knot-free, which is a graph structure that characterizes deadlocks in the OR model. We remark that deadlock detection can be done in polynomial time [70]. In this thesis, we mainly analyze the KFVD problem from a parameterized complexity point of view, which consists of determining whether G has a subset $S \subseteq V(G)$ of size at most k such that $G[V(G) \setminus S]$ contains no knot. There are three primary contributions:

1. A classical computational complexity analysis of KFVD where we show that:
   (a) KFVD is NP-hard when the input graph is either strongly connected or bipartite, planar with bounded out-degree;
   (b) KFVD can be solved in polynomial time when the input graph is subcubic;
   (c) KFVD with weights can be reduced into the unweighted version.
2. A fine-grained parameterized complexity analysis of KFVD where we show that:
   (a) KFVD is W[1]-hard when parameterized by the solution size, k;
   (b) KFVD is FPT when parameterized by k and the size of the largest strongly connected component;
   (c) KFVD is FPT when parameterized by the number of vertices with out-degree at most k;
   (d) Lower bounds on running time and kernelization based on the Exponential Time Hypothesis.
3. A parameterized complexity analysis of KFVD with graph width measures where we show that:
   (a) KFVD is W[1]-hard even if the input graph has longest directed path of length at most 5 and K-width equal to 2;
   (b) KFVD is FPT when parameterized by the size of the minimum directed feedback vertex set and:
      i. the K-width;

ii. the length of the longest directed path.

(c) KFVD is FPT when parameterized by clique-width of the graph;

(d) KFVD is FPT when parameterized by treewidth of the underlying graph.

1.2.1 Publications

We present a list of papers developed and published during the doctoral studies whose results are related to this thesis:

1. Bessy, S., Bougeret, M., Carneiro, A. D. A., Protti, F., Souza, U. S. Width Parameterizations for Knot-Free Vertex Deletion on Digraphs. *14th International Symposium on Parameterized and Exact Computation* (IPEC), 2019.

2. Carneiro, A. D. A., Protti, F., Souza, U. S. Deadlock Resolution in Wait-For Graphs by Vertex/Arc Deletion. *Journal of Combinatorial Optimization* (JOCO), 2019

3. Carneiro, A. D. A., Protti, F., Souza, U. S. Fine-Grained Parameterized Complexity Analysis of Knot-Free Vertex Deletion – A Deadlock Resolution Graph Problem. *The 23rd Annual International Computing and Combinatorics Conference* (COCOON), 2018.

4. Carneiro, A. D. A., Protti, F., Souza, U. S. Knot-Free Vertex Deletion Problem: Parameterized Complexity of a Deadlock Resolution Graph Problem *Latin American Workshop on Cliques in Graphs* (LAWCG), 2018.

5. Carneiro, A. D. A., Protti, F., Souza, U. S. A Parameterized Complexity Analysis of the Knot-Free Vertex Deletion Problem. *III ETC – Encontro de Teoria da Computação* (CSBC), 2018.

6. Carneiro, A. D. A., Protti, F., Souza, U. S. Deadlock Graph Problems Based on Deadlock Resolution. *The 23rd Annual International Computing and Combinatorics Conference* (COCOON), 2017.

7. Carneiro, A. D. A., Protti, F., Souza, U. S. Resolução de Deadlocks: Complexidade e Tratabilidade Parametrizada. *Simpósio Brasileiro de Pesquisa Operacional* (SBPO), 2017.

8. Carneiro, A. D. A., Protti, F., Souza, U. S. Deletion Graph Problems Based on Deadlock Resolution. *II ETC - Encontro de Teoria da Computação* (CSBC), 2017.

9. Barbosa, V. C., Carneiro, A. D. A., Protti, F., Souza, U. S. Deadlock Models in Distributed Computation: Foundations, Design, and Computational Complexity. *ACM Symposium on Applied Computing* (ACM SAC), 2016.

10. Carneiro, A. D. A., Protti, F., Souza, U. S. Algorithms for Deadlocks Resolution in Subcubic Graphs. *Latin American Workshop on Cliques in Graphs* (LAWCG), 2016.

11. Carneiro, A. D. A., Protti, F., Souza, U. S. Algoritmos para Resolução de Deadlocks em Grafos Subcúbicos. *Simpósio Brasileiro de Pesquisa Operacional* (SBPO), 2016.

12. Carneiro, A. D. A., Protti, F., Souza, U. S. Complexidade de Resolução de Deadlocks em Grafos de Espera de Sistemas Distribuídos. *Simpósio Brasileiro de Pesquisa Operacional* (SBPO), 2015.

Besides these works, we recently submitted two papers. the first entitled "Knot-free Vertex Deletion on Digraphs: A Parameterized Complexity Analysis" to *Algorithmica* journal and the second "Computational Complexity Aspects of Deadlock-Model Expressiveness" to the *Computational Complexity* journal.

1.3 Organization

The rest of the work is organized as follows.

In Chapter 2, we present the fundamental concepts and definitions. In particular, we define λ–DELETION($\mathbb{M}$) as a generic optimization problem for deadlock resolution, where $\lambda \in \{$VERTEX, ARC$\}$ indicates the type of deletion operation to be used in order to break all deadlocks, and $\mathbb{M} \in \{$AND, OR, X-OUT-OF-Y, AND-OR$\}$ is the deadlock model of the input wait-for graph G. We also present a review of parameterized complexity, where the concepts and some techniques of fixed-parameter tractability and intractability are discussed.

In Chapter 3, we present complexity results for all the eight combinatorial problems of the form λ–DELETION($\mathbb{M}$). First we point that VERTEX–DELETION(AND) and ARC–DELETION(AND) are equivalent to Directed Feedback Vertex Set and Directed Feedback Arc Set, respectively. Next, we present a computational complexity mapping considering the particular combination of deletion operations and deadlock models $\mathbb{M} \in \{$AND, OR, X-OUT-OF-Y, AND-OR$\}$ in simple directed graphs and for directed graphs with weighted vertices/arcs. A study of the complexity of KFVD in different graph classes is also done. We prove that the problem remains NP-hard even for strongly connected graphs and planar bipartite graphs with maximum degree four. Furthermore, we prove that for graphs with maximum degree three the problem can be solved in polynomial time. Finally, we show that λ–Deletion($\mathbb{M}'$) for deadlock models $\mathbb{M} \in \{\text{AND, OR}\}$ with weights can be reduced into the unweighted version.

In Chapter 4, we present a fine-grained parameterized complexity analysis of Vertex–Deletion(OR), so-called Knot-free Vertex Deletion (KFVD), where we first show that KFVD is W[1]-hard when parameterized by k, the size of the solution; next, we present two FPT-algorithms for KFVD considering different parameters. The first considers the size of the largest strongly connected component and the size of the solution. The second, the number of vertices with out-degree less or equal to the size of the solution. Furthermore, lower bounds based on SETH and ETH and some proofs of the infeasibility of polynomial kernelization are also presented.

In Chapter 5, we show that KFVD when parameterized by the treewidth of the underlying graph or by cliquewidth of the directed graph can be solved in FPT time. We also show a parameterized complexity analysis with several graph width measures, where we first improve the result that KFVD is W[1]-hard when parameterized by k, by showing that it remains W[1]-hard even if the input graph has a K-width at most 2 and the longest directed path has at most 5 vertices. Furthermore, we show that KFVD when parameterized by the directed feedback vertex set number together with either K-width or the longest directed path can be solved in FPT time. Besides that, lower bounds based on SETH and ETH and some proofs of the infeasibility of polynomial kernelization are also presented.

In Chapter 6, we present our final remarks and future work.

Chapter 2

Fundamental Concepts

This chapter presents a literature review with concepts and definitions pertinent to the problems addressed. The chapter is divided into three parts. In the first part, the main concepts and fundaments of distributed computation are presented. We introduce: the wait-for graphs, which are data structures used to represent distributed computing; the deadlock models, which are models that provide abstraction of the rules that govern the waiting of a process for its execution; the concepts and definitions of deadlock; the $\lambda-$DELETION($\mathbb{M}$) problems as a notation for deadlock resolution problems. In the second, the main notations and graph definitions are presented. Finally, we present a small review of the Parameterized Complexity theory.

2.1 Distributed Computation

According to [7], distributed-memory systems comprise a collection of processors interconnected in some fashion by a network of communication links, where processors do not physically share any memory, and the exchange of information among them must necessarily be accomplished by message passing over the network of communication links.

The architecture of a distributed computation can be represented by a graph $G = (V, E)$, where V is a set of processors that perform all distributed computing, and E is a set of communication channels that allows sharing resources through message passing. There are two main distributed computing architectures [6], synchronous and asynchronous, which differ mainly by time characterization.

Synchronous Architecture - It is mainly characterized by the existence of a global clock known to all processes. We can consider the clock as a step counter $s \geq 0$, and the message exchange between process neighbors of G occurs in a clock step; all tasks assigned to a process in time s are necessarily completed by starting step $s + 1$ [6].

Asynchronous Architecture - It is characterized by the absence of a global clock: each process has its own local clock, which is independent of the others. Message exchange between processes takes place in a finite time (guarantee of delivery) but indefinite. A process only performs some action upon receiving a message from a neighbor; Such action may include sending messages. At least one process must have some kind of spontaneous start or external intervention to start computing, as explained in [3, 27, 40].

The synchronous model offers some advantages due to the existence of a global clock, but usually, such a representation does not occur in practice; hence, from this point on, speaking of distributed computing or distributed system, we assume the asynchronous model.

In a distributed system, resource sharing is necessarily accomplished by message exchanges. The graph $G = (V, E)$ representing a distributed architecture is insufficient to represent a distributed computing in progress. A $v_i v_j$ edge exists in E if there is a direct communication channel between v_i and v_j; however, such abstraction does not provide us with any accurate information about message exchanges, which is fundamental to the study of deadlocks. Thus, we make use of wait-for graphs, which is presented in the next subsection.

In a distributed computation, a set of processes (vertices of the wait-for graph) is in *deadlock* if each process of the set is blocked, waiting for a response from another process of the same set. In other words, the processes cannot proceed with their execution because of necessary events, resources or a signal that only processes in the same set can provide. Deadlock is a *stable* property, in the sense that once it occurs in a global state Ψ of a distributed computation, it continues to hold for all the subsequent states to Ψ. Deadlock avoidance and deadlock resolution are fundamental problems in the study of distributed systems [3].

The main objective of this thesis is to study combinatorial problems related to deadlock resolution, in special, the deadlock resolution by process abortions (vertex deletion) problem in a distributed computation in the OR model. Our study is inspired by distributed systems; however, the issues addressed are not restricted to this scenario and and can be applied to any scenarios where deadlocks may occur. For this, we consider the waiting graphs of a distributed system, defined next.

2.1.1 Wait-for Graphs

Barbosa and Benevides [9] define wait-for graphs as structures of analysis and abstraction of distributed systems. These graphs, further detailed below, are dynamic structures, i.e., they change according to requests and responses of the system. We can disregard the nature of the dependency between the nodes of the network, that is, it will not be relevant for the purposes of this study what makes one node wait for another but if the wait occurs, because the study is directed to the existence of deadlocks, which as we have seen is a stable property. Once a deadlock occurs in a set of processes, only through external intervention (followed by the detection of deadlock) we may fix it.

We formally define the wait-for graph as a directed graph $G = (V, E)$, the vertex set V represents processes in a distributed computation, and the set E of directed arcs represents the wait conditions [9]. An arc exists in E directed away from $v_i \in V$ towards $v_j \in V$ if v_i is blocked, waiting for a signal from v_j. The graph G changes dynamically according to the deadlock model, as the computation progresses. In essence, the deadlock model specifies rules for vertices that are not *sinks* in G to become sinks [8]. (A sink is a vertex with out-degree zero). Wait-for graphs are the most common data structure to represent distributed computations, where the behavior of processes is determined by a set of prescribed rules (the *deadlock model* or *dependency model*).

The main deadlock models investigated so far in the literature are presented below.

a) AND MODEL – In the AND model, a process v_i can only become a sink when it receives a signal from *all* the processes in O_i, where O_i stands for the set of out-neighbors of v_i. This model applies to situations in which a conjunction of resources is needed by v_i [21, 69, 83].

b) OR MODEL – In this model, it suffices for a process v_i to become a sink to receive a signal from *at least one* of the processes in O_i. The OR model characterizes, for example, situations in which any single resource of a group (a disjunction of resources) is sufficient for v_i to proceed with its computation [21, 69, 74, 83].

c) X-OUT-OF-Y MODEL – There are two integers, x_i and y_i, associated with a process v_i. Also, $y_i = |O_i|$, meaning that process v_i is in principle waiting for a signal from every process in O_i. However, in order to be relieved from its wait state, it suffices for v_i to receive a signal from any x_i of those y_i processes. The X-Out-Of-Y model can then be applied to situations in which v_i starts by requiring access permissions above what it needs, and then withdraws the requests that may still be pending when the first x_i responses are received [20, 21].

d) AND-OR MODEL – There are $t_i \geq 1$ subsets of O_i associated with process v_i. These subsets are denoted by $O_i^1, \ldots, O_i^{t_i}$ and must be such that $O_i = O_i^1 \cup \cdots \cup O_i^{t_i}$. It suffices for a process v_i to become a sink to receive a signal from all processes in at least one of $O_i^1, \ldots, O_i^{t_i}$. For this reason, these t_i subsets of O_i are assumed to be such that no subset is contained in another. Situations that the AND-OR model characterizes are 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 [9, 21, 83].

Although distributed computations are dynamic, deadlock is a stable property; thus, 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 [7, 26].

2.1.2 Deadlocks

Informally, we say that a set of processes S is in a deadlock when every $i \in S$ is waiting for some condition to be fulfilled only by some action of one or more members of S itself. Classically, there are three main approaches to handling deadlocks [86]:

1. **Deadlock prevention:** It consists of identifying a condition C such that $\exists$ Deadlock $\rightarrow C$. Once C has been identified (which is required for a deadlock to occur), simply prohibiting the occurrence of C will **prevent** deadlocks. This approach is commonly achieved either by having a process secure all the needed resources simultaneously before it begins executing or by preempting processes which holds the needed resource.

2. **Deadlock avoidance:** It consists of when there are new requests for resources, a simulation (usually by blocking algorithm) is performed to verify the risk of deadlock. This approach to distributed systems, a resource is granted to a process if the resulting global system state is safe.

3. **Deadlock detection:** It consists of identifying a condition C such that $C \rightarrow \exists\ Deadkock$. Once C has been identified (which is sufficient for deadlock occurrence), simply check for C to detect **detect** the deadlocks. To handle deadlocks

using the approach of deadlock detection aproach lead to addressing two basic issues: detection of existing deadlocks and **resolution** of detected deadlocks.

Deadlock avoidance is considered an impractical strategy, requiring a lot of time and messages in order to know if a resource request is secure (it will not generate a deadlock). Thus, deadlock prevention and deadlock detection are the most viable approaches. Although prevention, avoidance, and detection of deadlocks have been widely studied in the literature, only a few studies have been dedicated to deadlock recovery [25, 43, 71, 88], most of them considering only the AND model. One of the reasons for this is that prevention and avoidance of deadlocks provide rules that are designed to ensure that a deadlock will never occur in a distributed computation. As pointed out in [88], deadlock prevention and avoidance strategies are conservative solutions, whereas deadlock detection is optimistic. Whenever the prevention and avoidance techniques are not applied, and deadlocks are detected, they must be broken through some intervention such as aborting one or more processes to break the circular wait condition causing the deadlock or preempting resources from one or more processes which are in deadlock. In this thesis, we consider such a scenario where deadlock was detected in a system and some minimum cost deadlock-breaking set must be found and removed from the system.

Although the basic principle of deadlock is model-independent, its characterization presents distinctions according to the deadlock model of the wait-for graph under analysis; Thus, we present the structures known in the literature that provide necessary and sufficient conditions for the existence of deadlock in a wait-for graph [8]:

AND model: A deadlock on a graph G in the AND model exists if and only if G contains a cycle.

OR model: A deadlock on a graph G in the OR model exists if and only if G contains a strongly connected component C of cardinality $|C| > 1$ where there are no paths from a C vertex to any $G[V \setminus C]$ vertex, that is, a strongly connected component with no exits and at least two vertices, called *knot*. Note that there are no paths from a knot to a sink.

AND-OR model: A deadlock on a graph G in the AND-OR model exists if and only if G contains a in b-knot: a strongly connected subgraph G' such that for each vertex $v_i \in V(G')$, at least one vertex of each subset of O_i also belongs to G'.

X-Out-Of-Y Model: A deadlock on a graph G in the X-Out-of-Y model exists if and only if G contains a (xy) *-knot*: a strongly connected G' subgraph such that for each vertex $v_i \in V(G')$, at least $(y_i - x_i + 1)$ nodes in the O_i set also belong to G'.

Once a deadlock is detected in distributed computing, some additional detection action, usually, some external intervention is required to resolve it (known as deadlock resolution). The deadlock detection and resolution algorithm always require that transactions should be aborted [25]. Notice that unnecessary aborts result in wasted system resources, thus, the aborts are usually more expensive than the waits. Finally, optimal concurrency requires that the number of aborted transactions be as few as possible.

Observation 2.1.1. *Sinks in G are vertices that do not depend on any vertex in G, that is, they are ready to perform their tasks.*

So, by Observation 2.1.1, it's easy to see that:

$$\nexists \text{ sink in } G \rightarrow \exists \textit{ Deadlock}.$$

2.1.3 $\lambda-$Deletion($\mathbb{M}$) Problems

We denote by λ–Deletion($\mathbb{M}$) a generic optimization problem for deadlock resolution, where λ indicates the type of deletion operation to be used in order to break all the deadlocks, and $\mathbb{M} \in$ {AND, OR, X-Out-Of-Y, AND-OR} is the deadlock model of the input wait-for graph G.

The types of deletion operations considered in this work are given below:

1. **Arc:** The intervention is given by arc removal. For a given graph G, Arc–Deletion($\mathbb{M}$) consists of finding the minimum number of arcs to be removed from G in order to make it deadlock-free. The removal of an arc can be viewed as the preemption of a resource.

2. **Vertex:** The intervention is given by vertex removal. For a given graph G, Vertex–Deletion($\mathbb{M}$) consists of finding the minimum number of vertices to be removed from G in order to make it deadlock-free. The removal of a vertex can be viewed as the abortion of one process.

The combination of intervention to be made and the deadlock model of the input graph gives eight possible combinatorial problems of the form λ–Deletion($\mathbb{M}$) (Table 2.1).

λ–Deletion($\mathbb{M}$)				
$\lambda \setminus \mathbb{M}$	AND	OR	AND-OR	X-Out-Of-Y
Arc	?	?	?	?
Vertex	?	?	?	?

Table 2.1: Computational complexity questions for λ–Deletion($\mathbb{M}$).

2.2 Graph Definitions and Notations

We use standard graph-theoretic and parameterized complexity notations and concepts, and any undefined notation can be found in [18, 41]. A directed graph $G = (V, E)$ consists of a set of vertices V with $n = |V|$ and a set of arcs E with $m = |E|$. We only consider loopless graph where for any $v \in V$, $vv \notin E$. Let $G[X]$ denote the subdigraph of G induced by the vertices in $X \subseteq V$. What we call here a directed path is a path without vertex repetition. Given a vertex v and a subset of vertices Z, we say that there is a path from v to Z if and only if there exists $z \in Z$ such that there is a vz-(directed) path. For $v \in V(G)$, let $D(v)$ denote the set of descendants of v in G , i.e. nodes that are reachable from v by a non-empty directed path. Given a set of vertices $C = \{v_1, v_2, \ldots, v_p\}$ of G, we define $D(C) = \bigcup_{i=1}^{p} D(v_i)$. Let $A(v_i)$ denote the set of ancestors of v_i in G, i.e., nodes that reach v_i through a non-empty directed path. We also define $A[v_i] = A(v_i) \cup \{v_i\}$, and given a set of vertices $C = \{v_1, v_2, \ldots, v_p\}$ of G, we define $A(C) = \bigcup_{i=1}^{p} A(v_i)$. For a vertex v of G, the out-neighborhood of v is denoted by $N^+(v) = \{u \mid vu \in E\}$, and given a set of vertices $C = \{v_1, v_2, \ldots, v_p\}$, we define $N^+(C) = \bigcup_{i=1}^{p} N^+(v_i) \setminus C$. The out-degree (resp., in-degree) of a vertex v is denoted by $deg^+(v)$ (resp., $deg^-(v)$). In addition, $\delta^+(G)$ (resp., $\delta^-(G)$) denotes the minimum out-degree (resp., in-degree) of a vertex in G. We refer to a Strongly Connected Component as an SCC. A knot in a directed graph G is an SCC Q of G with at least two vertices such that there is no arc uv of G with $u \in V(Q)$ and $v \notin V(Q)$. Finally, a sink (resp. a source) of G is a vertex with out-degree 0 (resp. in-degree 0). Given a subset of vertices S, we denote $G_S = G[S]$ and $\bar{S} = V \setminus S$. Thus, $G_{\bar{S}}$ denote the graph obtained by removing S.

2.3 Parameterized Complexity

From the beginning of the '70s, the development of the computational complexity theory, a wide number of problems have been categorized into "complexity classes" based on how fast they can be solved. The main complexity classes to take into account are the defined bellow.

Definition 2.3.1. *A problem* Π *belongs to the P class if and only if there is a deterministic algorithm* $\mathcal{A}$ *that for any instance* χ *of* Π *solves* χ *in polynomial time with respect to its size.*

Definition 2.3.2. *A problem* Π *belongs to the NP class if and only if for any yes-instance* χ *of* Π *with size* n*, there is a certificate (a string that certifies the "yes" response for the computation) that can be verified in polynomial time with respect to* n*.*

Definition 2.3.3. *A problem* Π' *is NP-hard if for any problem* $\Pi \in NP$*,* $\Pi \propto \Pi'$*, i.e., there exists an algorithm* $\mathcal{A}$ *that, given an instance* χ *of* Π *of size* n*,* $\mathcal{A}$ *constructs an instance* χ' *of* Π' *with size* $poly(n)$ *such that* χ' *is a yes-instance of* Π' *if and only if* χ *is a yes-instance of* Π*. If* $\Pi' \in NP$ *and it is NP-hard then* Π' *is NP-complete.*

The direct implications of Definitions 2.3.1 to 2.3.3 are that for any given problem Π, if $\Pi \in$ P, then, $\Pi \in$ NP. Furthermore, if any NP-hard problem Π can also be solved in polynomial time, then, every problem in NP can be solved in polynomial time. On the other hand, if an NP-complete problem Π cannot be solved in polynomial time, then, no NP-hard problem can be solved in polynomial time. From these observations, we get the most important unanswered question in computer science, "P = NP?" [46].

As pointed by Garey and Johnson [56], discovering that a problem is NP-complete is usually just the beginning of the work on that problem. The parameterized complexity theory was proposed by Downey and Fellows [27] as an alternative to deal with NP-hard problems. The parameterized complexity theory is a set of tools that can be used to better understand what aspects turn a problem hard to solve and, for instances where such aspects are fixed, it is possible to solve the problem efficiently. Next, we present two of the Karp's 21 NP-hard problems [67]:

VERTEX COVER (VC)
Instance: A graph $G = (V, E)$; a positive integer k.
Question: Does G have a vertex cover of size at most k? (A vertex cover is a set of vertices $S \subset V(G)$ such that $|S| \leq k$ and for every edge $uv \in E$, $u \in S$ or $v \in S$.)

INDEPENDENT SET (IS)
Instance: A graph $G = (V, E)$; a positive integer k.
Question: Do G have a vertex independent set of size at least k? (An independent set is a set of vertices $S \subset V(G)$ such that $|S| \geq k$ and $G[S]$ is a graph without any edge, that is, there are no two vertices adjacent in S.)

The parameterized complexity theory came from a simple but powerful idea from Rod Downey and Michael Fellows [46]. The combinatorial explosion that we have to deal with to solve an instance of an intractable problem somehow can be held in a "small structural characteristic" (called parameter) of this instance. These parameters are aspects of size, topology, shape, logical depth [50]. Usually is a numerical value that may depend on the input in an arbitrary way [52]. So, in parameterized complexity, the main interest is to design algorithms that may be exponential in relation to this small parameter but polynomial to the size of the input instance. Such an algorithm A formal definition of Fixed-Parameter Tractability (FPT) is presented below.

Definition 2.3.4. *A problem* Π *is Fixed-Parameter Treatable, if and only if, for any instance* $I = (\chi, k)$ *of* Π*, there is a algorithm* $\mathcal{A}$ *that correctly decides if* I *is a* yes- *or* no-*instance in time* $f(k).|I|^{O(1)}$*, where* k *is a parameter.*

It is helpful to see that, both VC and IS problems are NP-hard, and, from the classical computational complexity point of view, they are equivalent. In fact, both problems are closely related, notice that, for any input graph G, the dual of a maximum IS of G is a minimum VC of G. Furthermore, as pointed in [51], different problems, even as close as VC and IS, for the same parameter like the size of the solution k, contributes in two qualitatively different ways. It is known that VC is FPT when parameterized by the size of the solution k. On the other hand, IS is unlikely to be FPT when parameterized by the size of the solution k.

Both problems are solvable by a simple brute-force algorithm trying all subsets of size at most k that requires $O(n^{O(k)})$ time. *XP* is the class of parameterized problems that are solvable in time $O(n^{g(k)})$ for some function g. Proving the NP-hardness of a problem and having a parameter k bounded by a constant immediately forbids the existence of any XP (and thus algorithm) algorithm unless P = NP [39]. In this case, the parameterized problem is *para-NP-Hard*.

In the remainder of this section, we bring forward some useful tools of the parameterized complexity theory not only to design FPT algorithms but also, for some cases, to show that such an algorithm probably does not exist by establishing a parameterized intractability. Furthermore, the Exponential Time Hypothesis, a very handy conjecture to show lower bounds, is also presented.

2.3.1 Bounded Search Trees

This is one of the most commonly used tools in the design of fixed-parameter algorithms. The bounded search trees originate in the general idea of backtracking [41]. The algorithm tries to build a feasible solution to the problem by making a sequence of recursive branching decisions, typically by marking, removing, adding or labeling elements in a set of structure in each recursive call, reducing the size of the input instance until the answer is easily computable or even trivial.

Downey and Fellows point in [50] that the method of bounded search trees is fundamental to FPT algorithmic results in a variety of ways. A typical bounded search tree algorithm builds a search tree from the root where is the original instance and usually is decomposed into two parts:

i) Compute some search space of bounded size by a function of the parameter.

ii) Run some relatively efficient algorithm on each branch of the tree.

The computed search tree build in i) can be of exponential size in relation to the parameter, and the efficient algorithm ii) must be polynomial to the size of the input where the exploration of the search space is required. As pointed in [50], the worst-case may diverge significantly from the behavior of such algorithms on real datasets since probabilistic search space may not be fully explored and many branches ignored.

Next, we present an example application of the FPT algorithmic strategy of bounded search trees on a classical graph problem, the Vertex Cover.

k-VERTEX COVER (k-VC)
Instance: A graph $G = (V, E)$; a positive integer k.
Parameter: the size of the solution k.
Question: Does G have a vertex cover of size at most k? (A vertex cover is a set of vertices $S \subset V(G)$ such that $\lvert S \rvert \leq k$ and for every edge $uv \in E$, $u \in S$ or $v \in S$.)

We present next, a naive algorithm for k-VC using bounded search tree.

Lemma 2.3.5. *k-VC can be solved in FPT time.*

Proof. Let (G, k) be an instance of the k-VC. We start with S as an empty set. By the definition of the k-VC, it is clear that any VC S of the input graph G, each edge vu have at least one of its extremities in S. Notice that if a vertex is chosen to be in S, all of its neighbors are covered and we may remove both the vertex and its neighbors from G without losing. Therefore, we randomly chose a edge uv from E and recursively try both possibilities, $(G - u, k - 1)$ e $(G - v,\ k - 1)$. In each step, we reduce the parameter k by one. The algorithm stops if $k = 0$. If $E = \{\emptyset\}$, a vertex cover was found. The safety of the algorithm relies on the fact that branching is exhaustive, therefore, all possible VC of size at most k will be tested. □

2.3.2 Kernelization

Kernelization is a powerful technique commonly used to give FPT-algorithms for parameterized problems that mainly consist of, in polynomial time, transforming an input instance $I = (\chi, k)$ into a new instance $I' = (\chi', k')$ in such way that the size of I' is somehow bounded by the parameter k. A kernelization algorithm typically can be broken down in a series of small steps so-called reduction rules, usually taking advantage of specific features of the instance, which allow the safe reduction of the instance to an equivalent "smaller" instance [47]. Without loss of generality, kernelization can be seen as polynomial-time preprocessing with a guarantee. Thus, the technique has universal applicability, not only in the design of efficient FPT algorithms but also in the design of approximation and heuristic algorithms [58]. A formal definition is presented below.

Definition 2.3.6. Kernel: *Let Π be a parameterized problem, where $I = (\chi, k)$ is an instance of Π and k a parameter. We say that Π admits a kernel $I' = (\chi', k')$ if there is a algorithm $\mathcal{A}$ that, from (I, k), builds $I' = (\chi', k')$ (called problem kernel or just kernel) such that:*

i) *$k' \leq ck$, for a constant c;*

ii) *$|I| \leq g(k)$ for some function g;*

iii) *$I' = (\chi', k') \in \Pi$ is a equivalent instance of (I, k), i.e. $I' = (\chi', k')$ is a yes instance if and only if $I = (\chi, k)$ also is a yes instance.*

iv) *The algorithm $\mathcal{A}$ computes in polynomial time.*

We present next some reduction rules for the k-VC used to obtain a naive kernel.

Reduction rule 2.3.7. *Let (G, k) be a instance of k-VC. If there is a isolated vertex v, remove v from G obtaining a new instance $(G - v, k)$.*

The safety of Reduction rule 2.3.7 is trivial. Next, we explore the vertices degree.

Reduction rule 2.3.8. *Let (G, k) be a instance of k-VC. If there is a vertex v with $deg(v) > k$. Take v into the solution and remove v from G, thus, obtaining a new instance $(G - v, k - 1)$.*

The safety of Reduction rule 2.3.7 relies on the fact that, for every edge $uv \in E$, at least one of the two vertices u and v has to be in the vertex cover, then, if a vertex v has more than k neighbors, v must be in the solution.

Reduction rule 2.3.9. *Let $I = (G, k)$ be a instance of k-VC such that the Reduction rules 2.3.7 and 2.3.8 can no longer be appliyed. If at least one of the following conditions are satisfiyed, then I is a no instance.*

i) $k < 0$;

ii) G has more then $k^2 + k$ vertices;

iii) G has more then k^2 edges;

The safety of Reduction rule 2.3.9 relies on the fact that: i) if $k < 0$ then no vertex can be picked to be in the vertex cover; ii) if G has more then k^2+k vertices and $\Delta(G) \leq k$ (by Reduction rule 2.3.8), then, the k vertices to be choosen can cover at most k^2 neighbours. In iii), similar to ii), if G has more then k^2 edges and $\Delta(G) \leq k$ (by Reduction rule 2.3.8), then, the k vertices to be choosen can cover at most k^2 edges.

After Reduction rules 2.3.7 to 2.3.9, we obtain a quadratic kernel, since, either we find that the input instance is a no instance or that the size of the instance is at most $O(k^2)$.

Theorem 2.3.10. *The k-VC admits a kernel with $O(k^2)$ vertices and $O(k^2)$ edges.*

2.3.2.1 Lower Bounds on Kernelization

After applying the kernelization technique and getting a kernel of a parameterized problem, a natural question that arises is, how small this kernel can be? Of course, we would like the resulting kernel to be as small as possible, usually polynomial in relation to the parameter.

There are several positive results on the existence of kernels with polynomial size [49, 19, 2] or even linear [15, 16]. However, some parameterized problems probably do not support a kernel with polynomial size. In fact, in the past decade that the first results regarding the unfeasibility of kernels with polynomial size [46].

In order to show lower bounds on the kernel size, we use parameterized polynomial transformation (called PPT-reduction). Such a reduction is defined next.

Definition 2.3.11. PPT-reduction: *Let* $\Pi(k)$ *and* $\Pi'(k')$ *be parameterized problems where* $k' \leq g(k)$ *for some polynomial function* $g : \mathbb{N} \to \mathbb{N}$. *An PPT-reduction from* $\Pi(k)$ *to* $\Pi'(k')$ *is a reduction* R *such that:*

i) for all χ, *we have* $x \in \Pi(\chi)$ *if and only if* $R(\chi) \in \Pi'(k')$*;*

ii) R *is computable in polynomial time (in relation to* k*).*

2.3.3 Fixed-Parameter Intractability

From Section 2.3 to this point, the main goal was to show FPT algorithms techniques for parameterized problems. Such problems are fit in the FPT class. Besides the FPT class, Downey and Fellows defined the W hierarchy, a collection of computational complexity classes that accounts for the level of parameterized intractability [46]. To propper define the W hierarchy, we need some definitions.

Definition 2.3.12. ***[52] FPT-reduction:*** *Let* $\Pi(k)$ *and* $\Pi'(k')$ *be two parameterized problems where* $k' \leq f(k)$ *for some computable function* $f : \mathbb{N} \to \mathbb{N}$. *A FPT-reduction of* $\Pi(k)$ *to* $\Pi'(k')$ *is a reduction* R *such that:*

i) for all x, *we have* $x \in \Pi(k)$ *if and only if* $R(x) \in \Pi'(k')$*;*

ii) R *is computable in FPT time (in relation to* k*).*

It is important to notice that the FPT-reduction is transitive, that is, the FPT-reduction preserves the fixed-parameter tractability as follows. Given an FPT problem Π and a parameterized problem Π', if Π' is FPT reducible to Π, then, Π' is also FPT. On the other hand, if Π is not in FPT, then, Π' also is not in FPT [29].

To further discuss the W hierarchy, we describe a group of satisfiability problems on circuits of bounded depth.

Definition 2.3.13. *A Boolean circuit is of* mixed type *if it consists of circuits having gates of the following kinds.*

i) Small gates: not gates, and gates and or gates with bounded fan-in (usually assume that the bound on fan-in is 2 for and gates and or gates, and 1 for not gates).

ii) Large gates: And gates and Or gates with unrestricted fan-in.

The circuit can be represented by a directed AND-OR graph where a node represents a gate and each circuit has a single output. The weft of a circuit is the maximum number of large nodes on a path from an input node to the output node. We denote by $\mathcal{C}_{t,d}$ the class of circuits with weft at most t and depth at most d.

WEIGHTED CIRCUIT SATISFIABILITY (WCS[t])
Instance: A boolean circuit C with t large gates; a positive integer k.
Parameter: k.
Question: Does C have a satisfying assignment of weight k?

The parameterized problem WEIGHTED CNF FORMULA SATISFIABILITY (WCNF-SAT), consists of the pairs (F, k), where F is a boolean formula in the conjunctive normal form and k is the parameter such that the formula F has a satisfying assignment with weight k. The WEIGHTED CNF 3-SAT (WCNF-3SAT) problem is the WCNF-SAT problem restricted to instances where every clause of the formula F has at most three literals. Now we are ready to define the W-hierarchy.

Definition 2.3.14. *[29] The W hierarchy ($FPT \subseteq W[1] \subseteq W[2] \subseteq \cdots \subseteq W[t] \subseteq W[P]$) is a collection of computational complexity classes intuitively inspired in the* WEIGHTED CIRCUIT SATISFIABILITY *problem. The class $W[1]$ consists of all parameterized problems that are FPT-reducible to the problem* WCNF-3SAT. *For $t \geq 1$, a parameterized problem Π belongs to the class $W[t]$, if there is a FPT-reduction from Π to* WEIGHTED CIRCUIT SATISFIABILITY *on $\mathcal{C}_{t,d}$, for some $d \geq 1$.*

From definitions 2.3.12 and 2.3.14 we may define $W[t]$-hardness and completness (similar to Cook's theorem [31]) as follows. Given a parameterized problem Π, if WCS[t] (or a problem in $W[t]$) is PFT-reducible to Π, then, Π is $W[t]$-hard, additionaly, if Π is in $W[t]$, then, it is also $W[t]$-complete. Notice that for $t \geq 1$, if a problem Π is FPT, if a problem Π' is $W[t]$, if Π' is FPT-reducible to Π, then, FPT $= W[t]$, so, as in classical computational complexity there is the unanswered question "$P = NP$?", parameterized complexity theory has its own unanswered question, "$FPT = W[t]$?".

2.3.4 Lower Bounds on Exponential Time Hipothesis

The Exponential Time Hypothesis (ETH) and its strong variant, the Strong Exponential Time Hypothesis (SETH), are well-known and accepted conjectures that first appeared in [62] and are commonly used to prove lower bounds in parameterized computation. In the literature, several lower bounds have been found to many well-known problems, under such conjectures [41].

Conjecture 2.3.15. *[63, 72]* ***Exponential Time Hypothesis (ETH):*** *There is a positive real c such that 3-CNF-SAT cannot be solved in time $2^{cn}(n+m)^{O(1)}$, where n is the number of variables, and m is the number of clauses. In particular, 3-CNF-SAT cannot be solved in $2^{o(n)}(n+m)^{O(1)}$ time.*

Conjecture 2.3.15 is commonly used together with the Sparsification Lemma [63], meaning that 3-CNF-SAT cannot be solved in $2^{o(n+m)}(n+m)^{O(1)}$ time. In this work, without loss of generality, whenever we refer to ETH we mean to the latter version of the hypothesis. The Sparsification Lemma is presented below.

Lemma 2.3.16. *[63]* ***Sparsification Lemma*** *For all positive ϵ and positive r, there is a constant $K = K(\epsilon, r)$ such that any r-CNF formula F with n variables can be expressed as $F = \bigwedge_{i=1}^{r} c_i$, where $t \leq 2^{\epsilon n}$ and each c_i is a r-CNF formula with the same variable set as F and at most Kn clauses. Moreover, this disjunction can be computed by an algorithm running in time $O(2^{\epsilon n})$.*

Conjecture 2.3.17. *[63, 72]* ***A consequence of the Strong Exponential Time Hypothesis (SETH):*** *CNF-SAT cannot be solved in time $(2-\epsilon)^n(n+m)^{O(1)}$, where n is the number of variables, and m is the number of clauses.*

Conjecture 2.3.17 is an immediate consequence of the Strong Exponential Time Hypothesis (SETH) [64, 23].

In [63] a generalized reduction called Sub-Exponential Reduction Family (SERF) was introduced. A SERF reduction preserves sub-exponential time computation among search problems and their associated complexity parameters [22].

Definition 2.3.18. *[63, 66]* ***SERF-reduction.*** *Given two problems Π_1 and Π_2 with parameters κ_1 and κ_2, respectively, if Π_1 is SERF-reducible to Π_2, there is a Turing-reduction T_ϵ from Π_1 to Π_2 over all $\epsilon > 0$ with the following properties:*

i) the reduction $T_\epsilon(\chi)$ can be done in poly$(|\chi|) \times 2^{\epsilon\kappa_1(\chi)}$ time;

ii) if the reduction $T_\epsilon(\chi)$ outputs an input instance χ' then:

(a) $\kappa_2(\chi')$ is linearly bounded in $\kappa_1(\chi)$;

(b) the size of χ' is polynomially bounded in the size of χ.

Additional Notation

We denote by $dfv(G)$ the size of a minimum directed feedback vertex set of G. We generally use F to denote a directed feedback vertex set and by R the remaining subset, i.e., $R = V \setminus F$. The length of the longest directed path of G is denoted by $p(G)$. The Kenny-width [54] or K-width of G is denoted by $\kappa(G)$ and is the maximum number of distinct directed st-paths in G over all pairs of distinct vertices $s, t \in V(G)$, where two st-paths are distinct if and only if they do not use the same set of arcs. For any function g (like dfv, κ, p), $g(G)$ will be denoted simply by g when the considered graph G can be deduced from the context. In what follows, we denote by g–KFVD the KFVD problem parameterized by g ($g = k$ denotes the parameterization by the solution size). More concepts, notation, and definitions on parameterized complexity can be found in [41, 48, 52, 75].

Chapter 3

Classical Complexity

In this chapter, we present complexity results for all the eight combinatorial problems of the form λ–Deletion($\mathbb{M}$). The Vertex–Deletion(OR) problem receives a special analysis in Section 3.3. Finally, in Section 3.4, an analisys of the λ–Deletion($\mathbb{M}$) on weighted wait-for graphs in the AND and OR model is presented.

3.1 AND Model and Generalizations

To determine if there is a deadlock in a graph G in the AND model, it is necessary and sufficient to check the existence of cycles. Therefore, it is easy to see that Vertex–Deletion(AND) coincides with Directed Feedback Vertex Set (DFVS) and Arc–Deletion(AND) coincides with Directed Feedback Arc Set (DFAS), well-known problems proved to be NP-Hard in [67].

The AND-OR model is a generalization of the AND and OR models; therefore, every instance of a deadlock resolution problem for either the AND model or the OR model is also an instance for the AND-OR model. Also, the X-Out-Of-Y model also generalizes the AND and OR models. From this observation, it follows that:

Corollary 3.1.1. *For* $\mathbb{M} \in \{$AND-OR, X-Out-Of-Y$\}$*, it holds that:*

- Vertex–Deletion($\mathbb{M}$) *is NP-hard;*
- Arc–Deletion($\mathbb{M}$) *is NP-hard.*

3.2 OR Model

To determine if there is a deadlock in a wait-for graph G in the OR model, it is sufficient and necessary to check the existence of a *knot* [8]. Recall that a knot K is a strongly connected component (SCC) of order at least two where no vertex of K has an out-arc to a vertex that is not in K. Thus, in order to turn a wait-for graph deadlock-free, it is sufficient and necessary to turn the input graph into a knot-free graph. Therefore, we denote ARC–DELETION(OR) by Knot-Free Arc Deletion (KFAD) and VERTEX–DELETION(OR) by Knot-Free Vertex Deletion (KFVD).

3.2.1 Knot-Free Arc Deletion

The Knot-Free Arc Deletion problem is formally presented next.

KNOT-FREE ARC DELETION (KFAD)
Instance: A directed graph $G = (V, E)$; a subset $X \subseteq V$; and a positive integer k.
Question: Determine if G has a set $S \subset E(G)$ such that $|E| \leq k$ and $G[E \setminus S]$ is knot-free.

Since all strongly connected components on a graph G can be found through a topological ordering of G and a knot is a strongly connected component C with at least two vertices where there are no paths from a C vertex to any vertex in $G[V \setminus C]$, then, all the knots of a digraph can be identified in linear time as follows: first, find all the SCCs in linear time by running a depth-first search (Cormen et al. [32], pages 615-621); next, contract each SCC into a single vertex, obtaining an acyclic digraph H whose new sinks represent the knots of G.

Lemma 3.2.1. *Let G be a wait-for graph G in the OR model.*

(a) Let K be a knot. The minimum number of arcs to be removed in K to make it knot-free is $\delta^+(K)$.

(b) Let $\mathcal{K} = \{K_1, K_2, ..., K_p\}$ be the non-empty set of all the existing knots in G. The minimum number of arcs to be removed from G to make it knot-free is $\sum_{i=1}^{p} \delta^+(K_i)$.

Proof. (a) The key property to this proof is that any pair of vertices u, v in an SCC has a directed path to each other. Let v be a vertex with minimum out-degree in K. By removing all the out-arcs of v, v becomes a sink. Since K is an SCC, for every vertex $u \in V(K)$ all paths from u to v will remain intact; therefore, K will be knot-free. Since at least one sink must be created to make K knot-free, the minimum number of arcs to be removed of K is $\delta^+(K)$.

(b) By applying (a) repeatedly to each knot K_i in G, we solve all the knots with $\sum_{i=1}^{p} \delta^+(K_i)$ arc removals. Let $V' = \bigcup_{i=1}^{p} V(K_i)$. Since no arcs or vertices are changed outside $G[V(G)\backslash V']$, it is easy to see that no new knots will be created. Thus, $G[V(G)\backslash V']$ is knot-free. □

By Lemma 3.2.1 we can obtain in linear time a minimum set of arcs whose removal turns a given digraph G into a knot-free digraph.

Corollary 3.2.2. KNOT-FREE ARC DELETION *can be solved in linear time.*

Table 3.1 presents the computational complexities of the problems presented so far. The complexity analysis of VERTEX–DELETION(OR) is presented in next section.

		λ–DELETION($\mathbb{M}$)		
$\lambda \setminus \mathbb{M}$	AND	OR	AND-OR	X-Out-Of-Y
Arc	NP-H	P	NP-H	NP-H
Vertex	NP-H	?	NP-H	NP-H

Table 3.1: Partial scenario of the complexity of λ–DELETION($\mathbb{M}$).

3.3 Knot-Free Vertex Deletion

In this section, we show that KFVD is NP-hard. In addition, we analyze the problem for some particular graph classes and present a polynomial time algorithm for KFVD when the input graph is subcubic. The Knot-Free Vertex Deletion problem is formally presented next.

KNOT-FREE VERTEX DELETION (KFVD)
Instance: A directed graph $G = (V, E)$; and a positive integer k.
Question: Determine if G has a set $S \subset V(G)$ such that $|V| \leq k$ and $G[V \setminus S]$ is knot-free.

Lemma 3.3.1. KNOT-FREE VERTEX DELETION *is NP-hard.*

Proof. Let F be an instance of 3-SAT [56] with n variables and having at most 3 literals per clause. From F we build a graph $G_F = (V, E)$ which contains a set $S \subseteq V(G)$ such that $|S| = n$ and $G_F[V \backslash S]$ is knot-free if and only if F is satisfiable. The construction of G_F is described below:

1. For each variable x_i in F, create a directed cycle with two vertices ("variable cycle"), Tx_i and Fx_i, in G_F.

2. For each clause C_j in F create a directed cycle with three vertices ("clause cycle"), where each literal of C_j has a corresponding vertex in the cycle.

3. For each vertex v that corresponds to a literal of a clause C_j, create an arc from v to Tx_i if v represents the positive literal x_i, and create an arc from v to Fx_i if v represents the negative literal $\bar{x_i}$.

Figure 3.1 shows the graph G_F built from an instance F of 3-SAT.

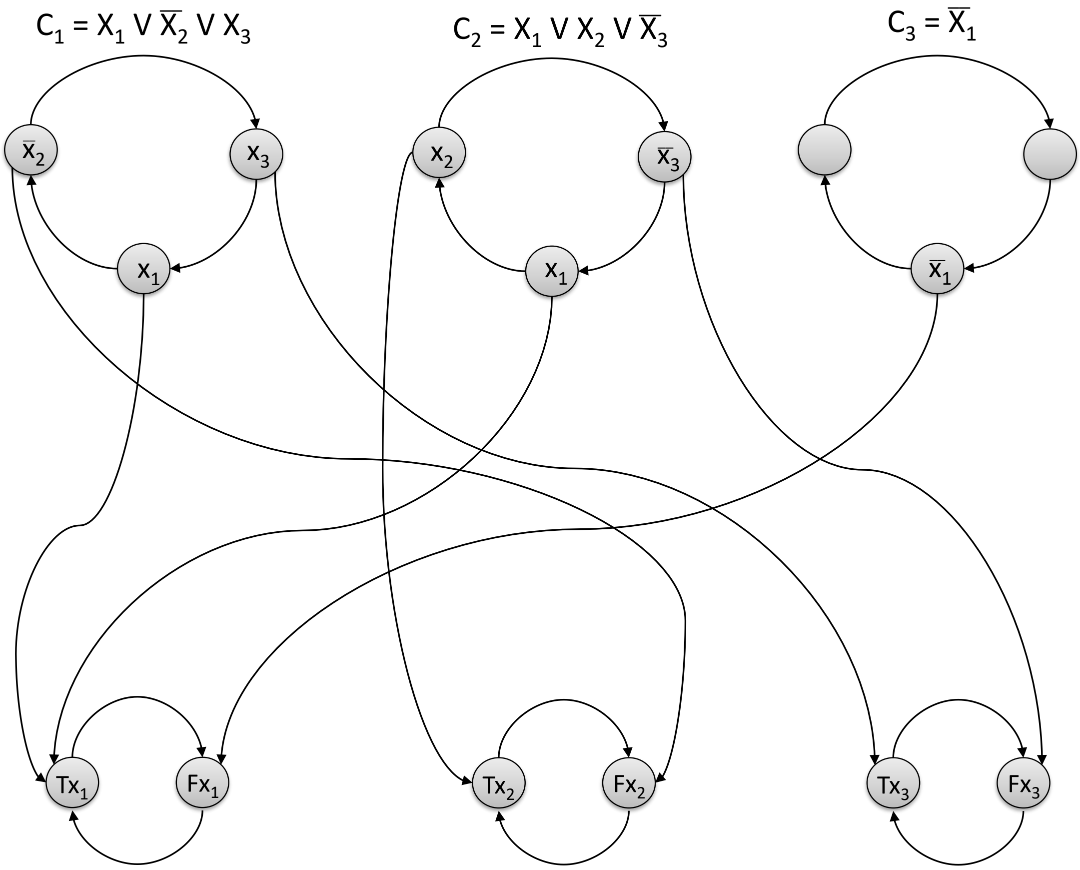

Figure 3.1: Graph G_F built from $F = (x_1 \vee \bar{x_2} \vee x_3) \wedge (x_1 \vee x_2 \vee \bar{x_3}) \wedge (\bar{x_1})$.

Suppose that F admits a truth assignment A. We can determine a set of vertices S with cardinality n such that $G_F[V \backslash S]$ is knot-free as follows. For each variable of F, select a vertex of G_F according to the assignment A such that the selected vertex represents the opposite value in A, i.e., if the variable x_i is true in A, Fx_i is included in S, otherwise Tx_i is included in S. Since each knot corresponds to a variable cycle, it is easy to see that $G_F[V \setminus S]$ has exactly n sinks. Therefore, since A satisfies F, at least one vertex corresponding to a literal in each clause cycle will have an arc towards a sink (vertex that matches the assignment). Thus $G_F[V \setminus S]$ will be knot-free.

Conversely, suppose that G_F contains a set S with cardinality n such that $G_F[V \setminus S]$ is knot-free. By construction, G_F contains n knots, each one associated with a variable of F. Hence, S has exactly one vertex per knot (one of Tx_i, Fx_i). As each cycle of $G_F[V \setminus S]$ corresponds to a clause of F, and $G_F[V \setminus S]$ is knot-free, each cycle of $G_F[V \setminus S]$ has at least one out-arc pointing to a sink. Thus, we can define a truth assignment A for F by setting $x_i = \mathit{true}$ if and only if $Tx_i \in \{V \setminus S\}$. Since at least one vertex corresponding to a literal in each clause cycle will have an arc towards a sink, we conclude that F is satisfiable. □

3.3.1 *Strongly Connected Graphs*

In general, a wait-for graph in the OR model can be viewed as a conglomerate of several strongly connected components. As observed in Subsection 3.2, the problems that can be solved in polynomial time have a characteristic in common: it suffices to solve every knot in G because no other SCC will become a knot after such removals. The next result deals with the natural question: "Can VERTEX–DELETION(OR) be solved in polynomial time when the input graph is strongly connected (i. e., G is a single knot)?".

Corollary 3.3.2. KNOT-FREE VERTEX DELETION *is NP-Hard even if G is strongly connected.*

Proof. Build a graph G_F as in Lemma 3.3.1, then add a universal vertex u (i.e., there are directed edges from u to all the other vertices, and vice-versa). Clearly, the resulting graph has a set of vertices S with cardinality $k+1$ such that $G_F[V \setminus S]$ is knot-free if and only if F is satisfiable. □

3.3.2 *Planar Bipartite Graphs with Bounded Degree*

Now we consider properties of the underlying undirected graph of G.

Since one of the most used architectures in distributed computation follows the user/server paradigm, an intuitively interesting graph class for distributed computation purposes are bipartite graphs. Planar graphs can also be interesting if physical settings must be considered; finally, bounded-degree graphs are very common in practice.

Theorem 3.3.3. KNOT-FREE VERTEX DELETION *remains NP-Hard even when the underlying undirected graph of* G *is bipartite, planar, and with maximum degree* 4.

Proof. Let F be an instance of PLANAR 3-SAT-AM3, where each variable has at most three occurrences with at least one positive and at least one negative. This problem is known to be NP-complete [81]. We show next that given a planar embedding H_F (the incidence graph corresponding to formula F[1]) we build an instance G_F of VERTEX-DELETION(OR) (a planar bipartite graph with $\Delta(G_f) = 4$) as follows:

- For each variable vertex v_{x_i} in H_F, create a directed cycle with two vertices ("variable cycle" X_i), Tx_i and Fx_i, in G_F.
- For each clause vertex w_{c_j} in H_F, create a directed cycle with six vertices ("clause cycle" C_j), where every two consecutive vertices in this cycle represent a literal in C_j, the first positive and the other negative.
- Considering H_F, choose an index $i \in \{1, \dots, n\}$ and suppose that x_i (the vertex representing the variable x_i) has degree at most three in H_F. This means that x_i occurs at most three times in F. Without loss of generality suppose that x_i occurs three times in F; therefore, in H_F, there are the edges (v_{x_i}, w_{c_j}), (v_{x_i}, w_{c_k}), and (v_{x_i}, w_{c_l}). Fig 3.2(a) shows vertex v_{x_i} with its incident edges in H_F. We then create edges in G_F by linking the clause cycles (corresponding to w_{c_j}, w_{c_k}, and w_{c_l}) to the variable cycle (corresponding to v_{x_i}). The added edges come from the first vertex corresponding to a literal if the literal is positive, and from the second otherwise. Observe in Fig 3.2 that the embedding of H_F is used as a "planar template" to guide the drawing of the edges leaving v_{x_i} in Fig 3.2(b). On the other hand, the added edges linking clause cycles and variable cycles can be seen from the clause cycle perspective. For each clause vertex w_{c_p}, $p \neq i$, formed by the occurrences of variables x_i, x_j, and x_k, there are in H_F the edges $(w_{c_p}, x_i), (w_{c_p}, x_j)$ and (w_{c_p}, x_k). Figure 3.3 illustrates the corresponding added edges in this perspective.

[1]The incidence graph of a CNF formula F is the bipartite graph $I(F)$ defined as follows: $V_1(I(F))$ consists of the variables of F and $V_2(I(F))$ consists of the clauses of F; a variable x and a clause C are adjacent if and only if x occurs (positively or negatively) in C.

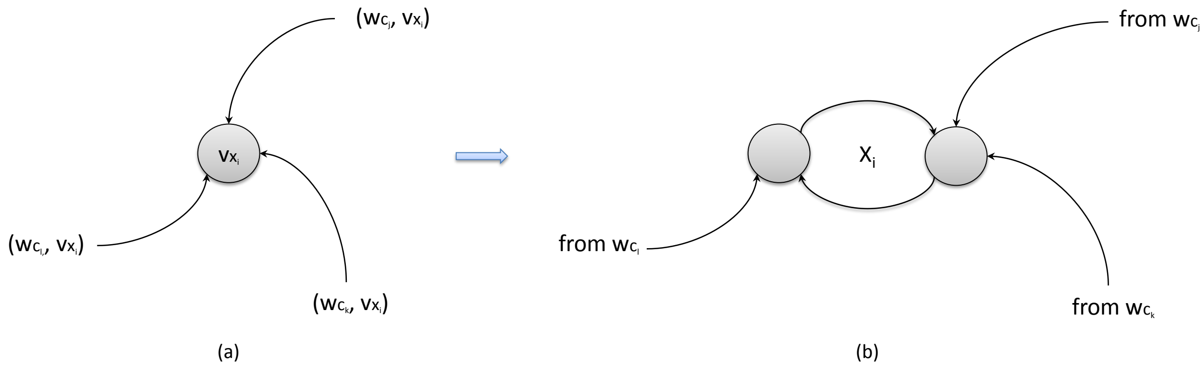

Figure 3.2: Variable cycle X_i built from a variable vertex v_{x_i} in H_F.

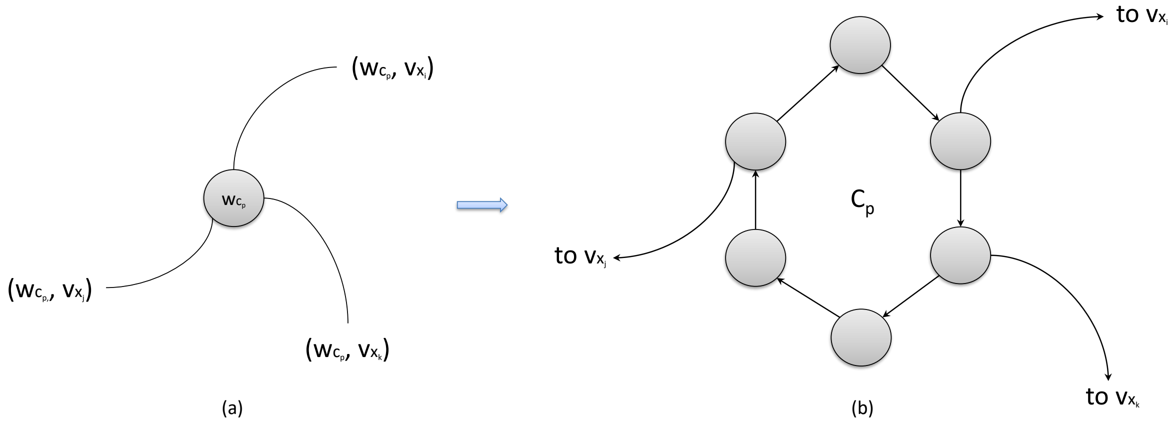

Figure 3.3: Clause cycle C_p built from a clause vertex w_{c_p} in H_F.

The bipartition of G_F can be easily deduced since no pair of vertices representing positive (resp. negative) literals are adjacent; moreover, in such cycles, if a vertex represents a positive literal and another vertex a negative literal then they are at an odd distance. In order to better understand the complete construction, the planar drawing, and the bipartition of G_F, Fig 3.4 shows a graph constructed from $F = (x_1) \wedge (x_2) \wedge (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3}) \wedge (\overline{x_3})$.

The rest of the proof follows directly from Lemma 3.3.1. □

3.3.3 Subcubic Graphs

Since Knot-Free Vertex Deletion remains NP-hard for graphs with $\Delta(G) = 4$, and is trivial for graphs with $\Delta(G) \leq 2$, an interesting question is to study the complexity of Vertex–Deletion(OR) when the underlying undirected graph of G has maximum degree three, i.e, is subcubic.

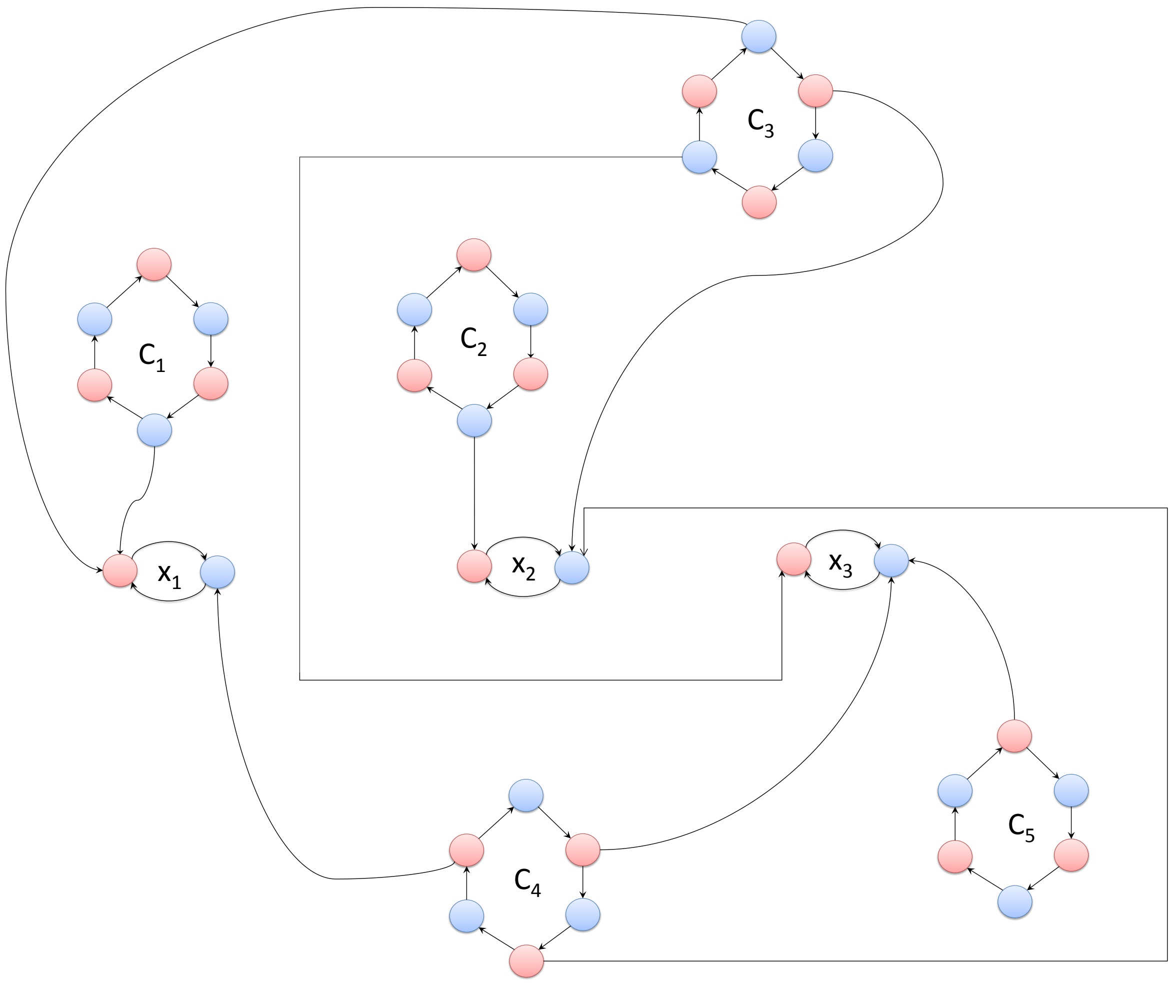

Figure 3.4: Graph G_F built from $F = (x_1) \wedge (x_2) \wedge (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3}) \wedge (\overline{x3})$.

To answer this question the first step is to phase out unnecessary vertices, i.e., vertices that never belong to any solution, such as sources and sinks.

Preprocessing. Let v_i be a sink or a source vertex in G; then, delete A_i from G, until G becomes source/sink free. Using depth-first search the preprocessing can be done in $O(n+m)$ time.

The safety of the preprocessing relies in the fact that sources will never be in a minimum VERTEX–DELETION(OR) set solution and that all vertices that reach a sink v_i (forming the set A_i) are already deadlock-free.

After the preprocessing, all vertices of G are in deadlock, and each one is classified into three types: **A** - with one in-arc and one out-arc; **B** - with one in-arc and two out-arcs; **C** - with two in-arcs and one out-arc.

The next step is to continuously analyze graph aspects in order to establish rules and procedures that may define specific vertices as part of an optimum solution. Thus, we iteratively build a partial solution contained in some optimum solution.

To break all the deadlocks in a graph G, it is necessary to destroy each knot in G. The removal of some vertices can destroy a knot; however, these removals may produce new knots that also need to be broken. Thus, our goal now is to identify for each knot a vertex that without loss of generality is part of an optimum solution.

Let W be an induced SCC of G. We can classify the vertices that are of type A in W into three sub-types (see Figure 3.5). A vertex is of sub-type $A.1$ if it is of type A in W, but of type C in the original graph, i.e., has an in-arc from another SCC; a vertex is of subtype $A.2$ if it is of the same type in both W and G; finally, a vertex is of subtype $A.3$ when it is of type A in W but of type B in G. Note that in a knot there will never be vertices of type $A.3$. It is worth noting that in a subcubic graph every knot vertex has at most one external neighbor (in-neighbor).

The following lemma presents an interesting relation between vertices of types B and C.

Lemma 3.3.4. *Let Q be a strongly connected subcubic graph. The number of vertices of type B in Q is equal to the number of vertices of type C.*

Proof. It is known that $\sum_{v_i \in V} deg^-(v_i) = \sum_{v_i \in V} deg^+(v_i)$ [85]. Since G is subcubic and strongly connected, G has only vertices A, B, and C . Note that a vertex of type A has one in-arc

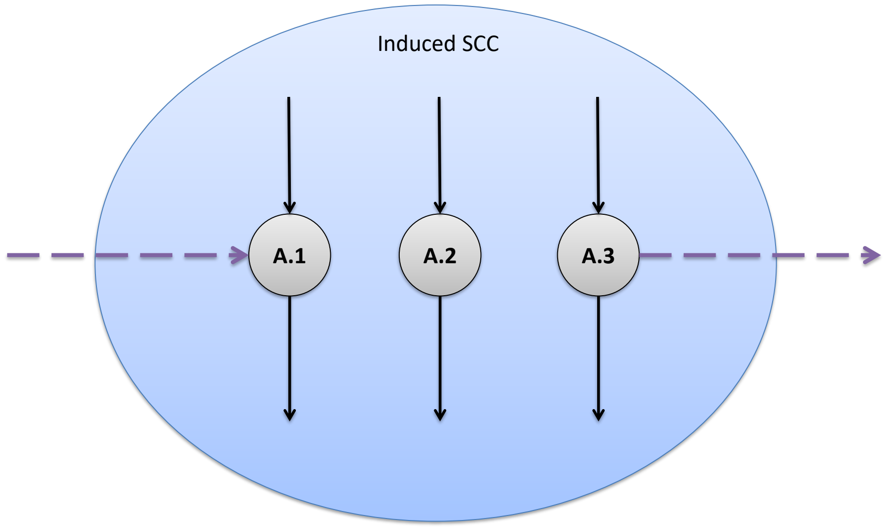

Figure 3.5: Subtypes of vertices A in an SCC.

and one out-arc; therefore it is easy to see that in order to maintain the same number of in- and out-arcs of G, G must have an equal number of vertices of types B and C. □

At this point, we can identify some vertices of an optimal solution.

Theorem 3.3.5. *Let G be a subcubic graph and Q be a knot in G. If Q contains a vertex of type B, C, or $A.2$, then G has an optimal solution S for* VERTEX–DELETION(OR) *which contains exactly one vertex $v_i \in V(Q)$.*

Proof.
(a) Suppose first that Q contains a vertex of type B. Since Q is strongly connected, any sink arising from a vertex removal will break Q. Since a vertex of type B has no neighbor outside Q (otherwise Q would not be a knot), removing it does not create a knot in $G - V(Q)$. Thus, given a vertex v_i of type B in Q, the removal of v_i will not turn its in-neighbor w_i into a sink only if w_i is also of type B. In this case, we repeat the same process for w_i. From Lemma 3.3.4, eventually, we find a vertex v_j of type B whose in-neighbor w_j is not of type B, otherwise, Q would be composed only by vertices of type B.

(b) Suppose now that Q contains a vertex of type C. The proof for this case follows directly from Lemma 3.3.4 and (a).

(c) Finally, suppose that v_i is a vertex of type $A.2$ in Q. Since such a vertex has no neighbors outside Q, removing it does not create a knot in $G - V(Q)$. In addition, the removal of v_i does not break Q only if its in-neighbor w_i is of type B. In this case, we can apply (a). □

Corollary 3.3.6. *The vertex in Theorem 3.3.5 can be found in linear time.*

Corollary 3.3.7. *Let Q be a knot of a subcubic graph G. If there is no vertex v_i in Q such that $G - v_i$ has fewer knots than G, then Q is a cycle composed only by vertices of type $A.1$.*

Proof. By Theorem 3.3.5, Q clearly contains no vertices of types B, C, or $A.2$. Furthermore, it cannot have any vertices of type $A.3$. Thus, Q is a cycle of vertices of type $A.1$. □

Figure 3.6 shows a graph with a knot (SCC in red) composed only by vertices of type $A.1$.

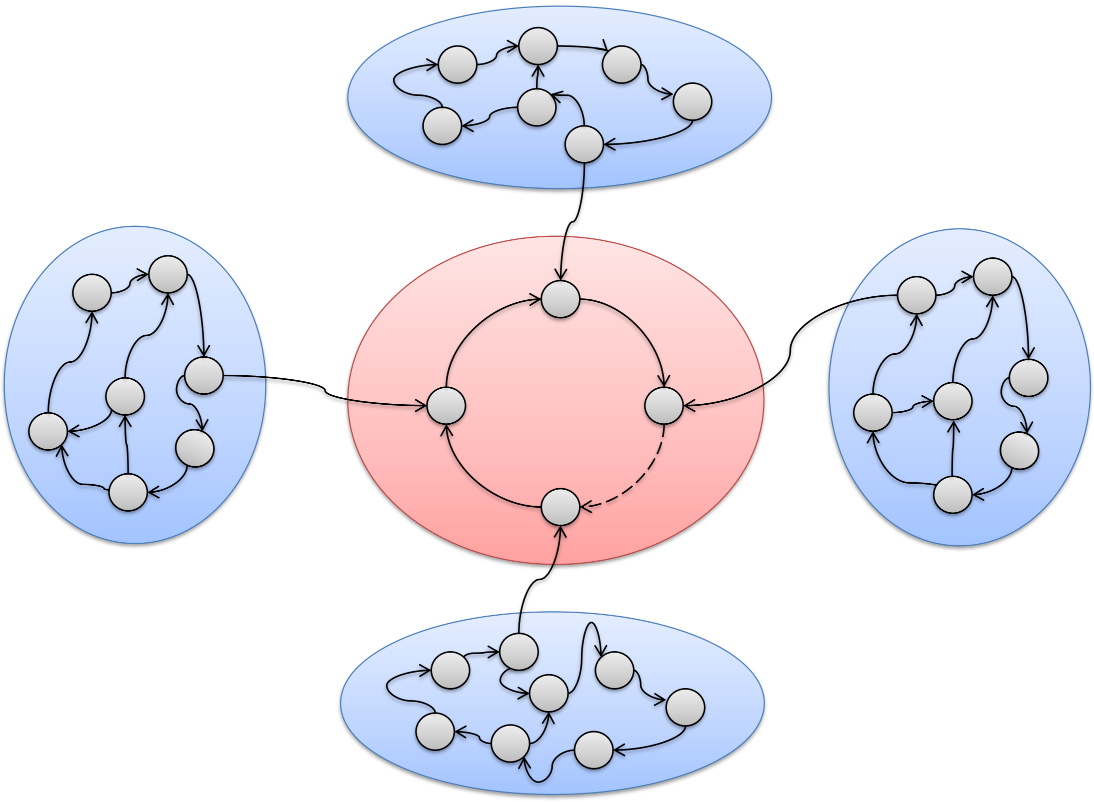

Figure 3.6: A graph with a knot composed only by vertices of type $A.1$.

Now, we can determine lower and upper bounds for the problem.

Lemma 3.3.8. *Let S be an optimal solution of a subcubic instance G of* Knot-Free Vertex Deletion. *Then it holds that $k \leq |S| \leq 2k$, where k is the number of knots of G.*

Proof. For a given subcubic graph G containing k knots, G needs at least k vertex deletions (one in each knot) in order to break all the knots. Conversely, in the worst case, all the knots are composed only by vertices of type $A.1$ (Corollary 3.3.7) and we show next that at most $2k$ removals are required to make G deadlock-free. If a knot Q_i has at least one vertex that is not of type $A.1$, apply Theorem 3.3.5. Since at least one vertex deletion is needed in each knot and all remaining knots are composed only by vertices of type $A.1$, for each knot Q_i simply remove any vertex v_i. Observe that the deletion of a vertex of type $A.1$ in a knot creates at most one new knot Q'_i (the SCC of the in-neighbor w_i of the deleted vertex v_i); furthermore, w_i becomes a vertex of type $A.2$ after the deletion of v_i. Finally, considering that the new knot Q'_i has at least one vertex of type $A.2$, Q'_i can be solved with only one vertex deletion (Theorem 3.3.5). □

Corollary 3.3.9. Knot-Free Vertex Deletion *in subcubic graphs can be 2-approximated in linear time.*

Proof. Follows from Theorem 3.3.5 and Lemma 3.3.8. □

3.3.3.1 A Polynomial Time Algorithm.

In order to obtain an optimum solution in polynomial time, there are some significant considerations to be made regarding the remaining graph.

Lemma 3.3.10. *Let G be a subcubic instance of* Knot-Free Vertex Deletion. *Let $\mathcal{C} = \{C^1, C^2, \ldots, C^j\}$ be a set of non-knot SCCs of G, where for each $C_i \in \mathcal{C}$ there is a directed path from C^i to C^{i+1} and from C^j to a knot Q. Then:*

(a) No vertex in $C^1, C^2, \ldots, C^{j-1}$ is part of an optimal solution.

(b) If C^j has two or more out-arcs pointing to Q or there is some $i < j$ for which C^i is directly connected to Q then there is an optimal solution S such that $V(Q) \cap S = \{v_i\}$, and such a vertex can be found in linear time.

Proof.

(a) Note that each C^i, $1 \leq i \leq j-1$, has a path to C_j; therefore, in the worst case two removals will take place (one in Q and other in C^j), and thus all the SCCs $C^1, C^2, \ldots, C^{j-1}$ will have a path to a sink.

(b) If C^j has two or more out-arcs pointing to Q, without loss of generality we can remove an in-arc from C^j to a vertex of type $A.1$ in Q. In so doing, such a vertex becomes a vertex of type A.2 and Theorem 3.3.5 can be applied. Analogously, if there is some $i < j$ where C^i is directly connected to Q, we can also remove an in-arc from C^i to a vertex of type $A.1$ in Q and apply Theorem 3.3.5. □

At this point, we are able to apply the first steps of our algorithm.

First steps of the algorithm. (i) Remove any SCC that is pointing to another non-knot SCC; (ii) If a non-knot SCC has at least two arcs pointing to the same knot then remove all such arcs but one; (iii) For each knot Q having vertices of type B, C, or $A.2$, find a vertex v_i that, without loss of generality, is in an optimal solution and remove it; (iv) Remove all vertices that are no longer in deadlock.

The correctness of the above steps follows from Theorem 3.3.5 and Lemma 3.3.10. Observe also that such routines can be performed in $O(n + m)$ time.

Now, consider the graph G obtained after applying the above steps. The knots of G are directed cycles composed by vertices of type $A.1$ (see Corollary 3.3.7), and each non-knot SCC of G is at distance one of a knot. At this point, any vertex v removed from a knot Q will break it, but potentially creates a new knot from the SCC W that has an out-arc to v. However, such new knot W will have a vertex of type $A.2$; therefore, W can be solved by the removal of another vertex w in W, called *solver vertex*.

Our final step is to minimize the number of solver vertices that actually need to be removed in order to make the graph knot-free. To achieve this purpose, we will consider the bipartite graph $B = (K \cup C,\ E)$, where K is the set of vertices representing contracted knots, and C is the set of vertices representing contracted non-knot SCCs. Note that each arc from a vertex c_j in C to another vertex k_i in K represents a connection from a vertex w of the SCC represented by c_j to a vertex v of the knot represented by k_i, and indicates that w becomes a vertex type of $A.2$ after the removal of v, which guarantees the existence of a solver vertex, that may or may not be used. Therefore, we seek a set M' of arcs in B such that:

1. Each vertex in K is adjacent to at most one arc of M'. (This arc indicates the vertex of the knot to be removed.)

2. Each vertex in C has at least one arc that does not belong to M'. (This arc indicates a path to a sink, a broken knot in K; thus, the SCC is deadlocked without removing internal vertices.)

3. M' is maximum.

From the set M' we can obtain an optimal solution S such that $G[V \setminus S]$ is knot-free (where G is prior to the contraction). In fact, M' indicates the maximum number of knots that can be broken without generating new knots, as well as the number of solver vertices needed. A solution S can be built as follows: for each vertex k_i in K, saturated by M', include in S the associated knot vertex in G; for every vertex $k_i \in K$ such that k_i is not saturated by M', choose an arc $e \in E(B)$ and include in S the knot vertex of G associated with e; then, for the SCC C of G indicated by the arc e, include in S a solver vertex. Figure 3.7 shows a set M' for a graph $B = (K \cup C, E)$. Each unsaturated vertex in K (vertex in red) suggests one solver vertex in C that needs to be in S.

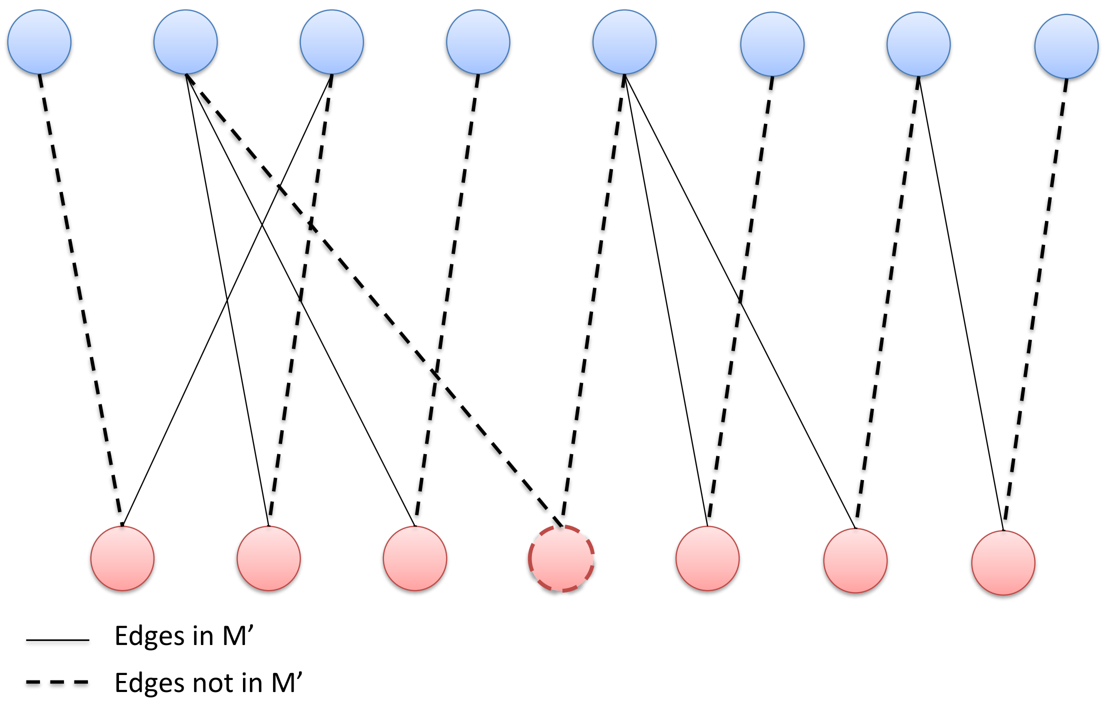

Figure 3.7: Sample of set M' saturating $t-1$ vertices in a bipartite graph $B = (K \cup C, E)$. Vertices in K are red, and vertices in C are blue. Solid lines represent edges of M', while dashed lines represent edges not in M'. The highlighted vertex in red is not saturated by M'.

The set M' can be obtained by using the concept of (f, g)-*semi-matching*. An (f, g)-semi-matching is a generalization of the concept of semi-matching presented in [17]. Let $f : K \to \mathbb{N}$ and $g : C \to \mathbb{N}$ be functions. An (f, g)-semi-matching in a bipartite graph $B = (K \cup C, E)$ is a set of arcs $E' \subseteq E$ such that each vertex $k_i \in K$ is incident with at most $f(k_i)$ arcs of E', and each vertex $c_j \in C$ is incident with at most $g(c_j)$ arcs of E'.

In fact, M' is an (f, g)-semi-matching where, for every $k_i \in K$, $f(k_i) = 1$, and for every $c_j \in C$, $g(c_j) = deg(c_j) - 1$.

Lemma 3.3.11. [17, 53, 68] *Given a bipartite graph $B = (K \cup C,\ E)$ and two functions $f : K \to \mathbb{N}$ and $g : C \to \mathbb{N}$, finding a maximum (f, g)-semi-matching of B can be done in polynomial time, and a maximum $(1, g)$-semi-matching of B can be found in $O(m\sqrt{n})$ time.*

The algorithms for (f, g)-semi-matchings use similar ideas to those used in the well-known algorithm by Hopcroft and Karp [60].

At this point, we have all the elements to answer the question raised on the complexity of Vertex-Deletion(OR).

Theorem 3.3.12. Knot-Free Vertex Deletion *restricted to subcubic graphs is solvable in $O(m\sqrt{n})$ time.*

Proof. First, we must perform all the preprocessing previously presented. Then, we build the equivalent bipartite graph B. Given the bipartite graph B, the set of arcs M' is an (f, g)-semi-matching where, for every $k_i \in K$, $f(k_i) = 1$, and for every $c_j \in C$, $g(c_j) = deg(c_j) - 1$. The $(1, g)$-semi-matching can be computed in $O(m\sqrt{n})$ time.

If $|M'| = |K|$, then for every knot a vertex in the original graph G is chosen to be deleted (a vertex in a knot $k_i \in K$ saturated by M'), and each component c_j of C is released through a solved knot (by some arc that is not in M'). Thus, the arcs of M' induce an optimum set of vertices to be removed.

Suppose a maximum M' such that $|M'| = |K| - q$ for some $q \geq 1$. Since each vertex v in K sees at most one arc in M' and $|M'| = |K| - q$, q vertices of K are not saturated by the $(1, g)$-semi-matching . The number of vertices not saturated by M' is equal to the number of components that are turned into knots (after solving all knots not saturated by M') and the number of solvers that need to be removed. Since M' is maximum and $|M'| = |K| - q$, we have that q is minimum, and from M' we build a set S with $|S| = |K| + q$ vertices such that $G[V(G) \setminus S]$ is knot-free as previously explained: for each vertex k_i in K, saturated by M', include in S the associated knot vertex in G; for every vertex $k_i \in K$ such that k_i is not saturated by M', choose an arc $e \in E(B)$ and include in S the knot vertex of G associated with e; then, for the SCC C of G indicated by the arc e, include in S a solver vertex, by Theorem 3.3.5. See Figure 3.7.

Conversely, if we have a set S of vertices of cardinality $|K| + q$ such that $G[V(G) \setminus S]$ is knot-free, S induces a set M' of arcs in the bipartite graph B that saturates $|K|-q$ vertices of K using arcs that also saturate the SCCs in C that do not have solver vertices in S. Therefore, there is a $(1, g)$-semi-matching for the bipartite graph B of size $|K|-q$. $\square$

3.4 Weighted-λ-*Deletion*($\mathbb{M}$)

Priority-based distributed computations have multiple applications like Job Scheduling [4], Resource Allocation [73], among others. Wait-for graphs with weights in a distributed computation model can express de degree of priority of processes or requisitions. We denote by $P(x)$ the weight of a vertex/arc x and $\varrho^+(v)$ the set of out-arcs of v. i.e. there is a arc $e_i = (v, v_i) \in \varrho(v)\ \forall\ v_i \in N^+(v)$. In this section we show that λ-*Deletion*($\mathbb{M}$) on weighted directed graphs (W-λ-*Deletion*($\mathbb{M}$)) is equivalent to the λ-*Deletion*($\mathbb{M}$) problems on non-weighted directed graphs.

Formally, we define the deletion operations of W-λ-*Deletion*($\mathbb{M}$) as follows:

1. **Arc:** The intervention is given by arc removal. For a given weighted directed graph G, W-Arc–Deletion($\mathbb{M}$) consists of finding a set of arcs to be removed from G with minimum combined cost in order to make it deadlock-free.
2. **Vertex:** The intervention is given by vertex removal. For a given graph G, Vertex–Deletion($\mathbb{M}$) consists of finding a set of vertices with minimum combined cost to be removed from G in order to make it deadlock-free.

3.4.1 Weighted AND Model

As pointed in Section 3.1, to determine if there is a deadlock in a graph G in the AND model, it is necessary and sufficient to check the existence of cycles. Therefore, it is easy to see that W–Vertex–Deletion(AND) coincides with Weighted Directed Feedback Vertex Set (WDFVS) and W–Arc–Deletion(AND) coincides with Weighted Directed Feedback Arc Set (WDFAS). Next, we show that in fact that Vertex–Deletion(AND) is reducible to W–Vertex–Deletion(AND). First, we show for the vertex version (the vertices have weights) and second we show for the arc version (The arcs have weights).

Lemma 3.4.1. *Let G be an instance of* W–Arc–Deletion(AND). *We can transform G into an equivalent instance G' of* Arc–Deletion(AND) *in polynomial time if for any arc $e \in E(G)$, $P(e) \leq poly(n)$.*

Proof. We show that W–ARC–DELETION(AND) $\propto$ ARC–DELETION(AND). We build G' from G as follows. Set $G' = G$. Let $x = P(e_i)-1$, if $x \geq 1$, for each arc $e_i = (v,u) \in \varrho^+(v)$ create x artificial vertices $v_u^1, ..., v_u^x$. Also, create an arc from v to each v_u^i and from each v_u^i to u in G' (see Figure 3.8).

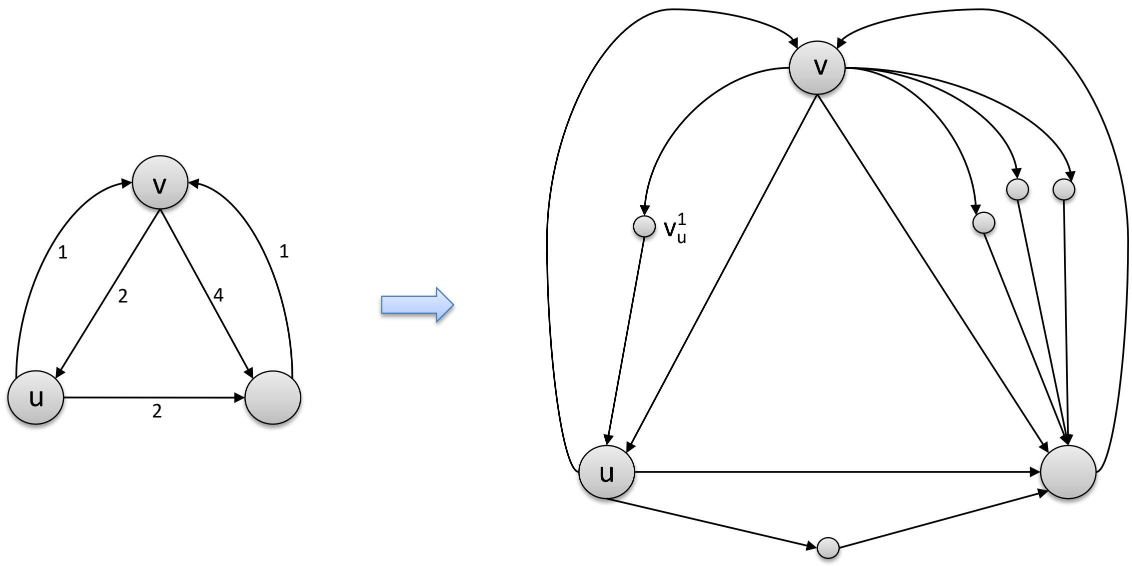

Figure 3.8: Example of an instance G of W–ARC–DELETION(AND) with weights in the arcs and the correspondent gadget in of an instance G of ARC–DELETION(AND).

Suppose that G has a set S of arcs such that $\sum_{e_i \in \varrho^+(v)} P(e_i) = k$ and $G[E(G) \setminus S]$ is knot-free. We build a set S' such that $G'[V(G') \setminus S']$ is knot-free. For each arc $e_i = (v,u)$ in S we put in S' the additional $P(e_i)$ arcs ($P(e_i) - 1$ arcs from v_i to the x artificial vertices $v_u^1, ..., v_u^x$ and one from v to u). Since $G[V(G) \setminus S]$ is cycle-free, and all additional out-arcs from v created to each deleted e_i in G is also in S', $G'[V(G') \setminus S']$ is knot-free and S' has size exactly k.

Conversely, Suppose that G' has a set of arcs S' with size k and $G'[V(G') \setminus S']$ is cycle-free. We create a set S such that $\sum_{e_i \in \varrho^+(v)} P(e_i) \leq k$ and $G'[V(G') \setminus S']$ is knot-free as follows. For each arc $e' = (v,u)$ in S' we put $e = (v,u)$ in G, i.e. we just ignore the arcs that goes to/from a artificial vertex v_u^i. Notice that even if we delete all de ignored edges in the graph, if there is an arc in S' that goes to/from a artificial vertex v_u^i, then, In order to get a knot-free graph, the u, v arc must be in S'. So, if the arc (v,u) is in S', S' must have one other arc for each artifitial v_u^i that either goes from v or to u. Finally, since G is equal G' without the artificial vertices and $G[E(G) \setminus S']$ is cycle-free, then, $G[E(G) \setminus S]$ is also cycle-free. $\square$

Next is presented wait-for graphs with weighted vertices, i.e, a distributed computation where the processes have priorities. We show next that W–VERTEX–DELETION(AND) is reducible to VERTEX–DELETION(AND), thus solvable in polynomial time.

Lemma 3.4.2. *Let G be an instance of* W–VERTEX–DELETION(AND)*. We can transform G into an equivalent instance G' of* VERTEX–DELETION(AND) *in polynomial time if for any vertex $v \in V(G)$, $P(v) \leq poly(n)$.*

Proof. We show that W–VERTEX–DELETION(AND) $\propto$ VERTEX–DELETION(AND). We build G' from G as follows. For each vertex $v_i \in V(G)$ with weight $x = P(v_i)$ we create $v_i^1, ..., v_i^x$ vertices in G'. For each arc u, v in G, create an arc from each u_i^j to each v_l^z in G' (see Figure 3.9).

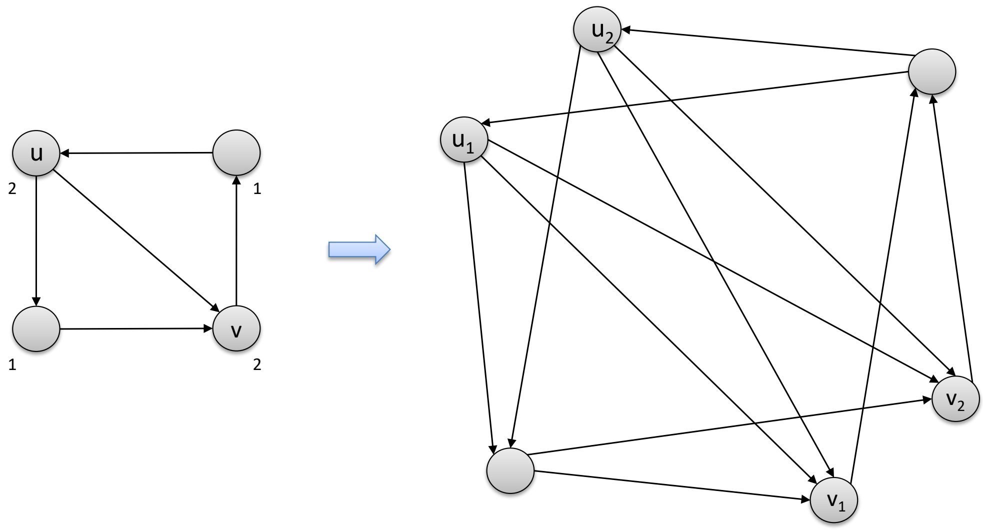

Figure 3.9: Example of an instance G of W–ARC–DELETION(AND) with weights in the vertices and the correspondent gadget in of an instance G of ARC–DELETION(AND).

Suppose that G has a set S such that $\sum_{v_i \in S} P(v_i) = k$ and $G[V(G) \setminus S]$ is cycle-free. We create a set S' of size k such that $G'[V(G') \setminus S']$ is cycle-free. For each vertex $v \in S$ put all the v_i^j vertices of G' in S'. Since by construction each vertex v_i^j have the same in- and out-neighbors, clearly $G'[V(G') \setminus S']$ is cycle-free. As there are $P(v_i)$ vertices in G', $|S'| = k$.

Conversely, suppose that G' has a minimum set S' of size k such that $G'[V(G') \setminus S']$ is knot-free. We create a set S such that $\sum_{v_i \in S} P(v_i) = k$ such that $G[V(G) \setminus S]$ is knot-free as follows. If there is a vertex $v_i^j \in S'$, put v in S. Since any pair of vertices v_i^j and v_i^l have the same in- and out-neighbors, if v_i^j is in S, v_i^l also has to be in S. Therefore, $G[V(G) \setminus S]$ is knot-free and clearly $\sum_{v_i \in S} P(v_i) = k$. $\square$

3.4.2 Weighted OR Model

We first analyze wait-for graphs with weighted arcs, we call WEIGHTED KNOT FREE ARC DELETION (WKFAD) the weighted version of KFAD. We show next that PKFAD is reducible to KFAD, thus, solvable in polynomial time.

Lemma 3.4.3. *Let G be an instance of* PKFAD. *We can transform G in polynomial time into an instance G' of* KFAD *if for any arc $e \in E(G)$, $P(e) \leq poly(n)$.*

Proof. We show that WKFAD $\propto$ KFAD. We build G' from G as follows. Set $G' = G$. For each arc $e_i = (v, u) \in \varrho^+(v)$, if $P(e_i) \geq 2$, create a directed complete subgraph Q_{e_i} of size $\sum_{e_j \in \varrho^+(v)} P(e_j) + 2$ in G', and create $P(e_i) - 1$ arcs from v to Q_{e_i} and one arc from Q_{e_i} to u in G' (see Figure 3.10).

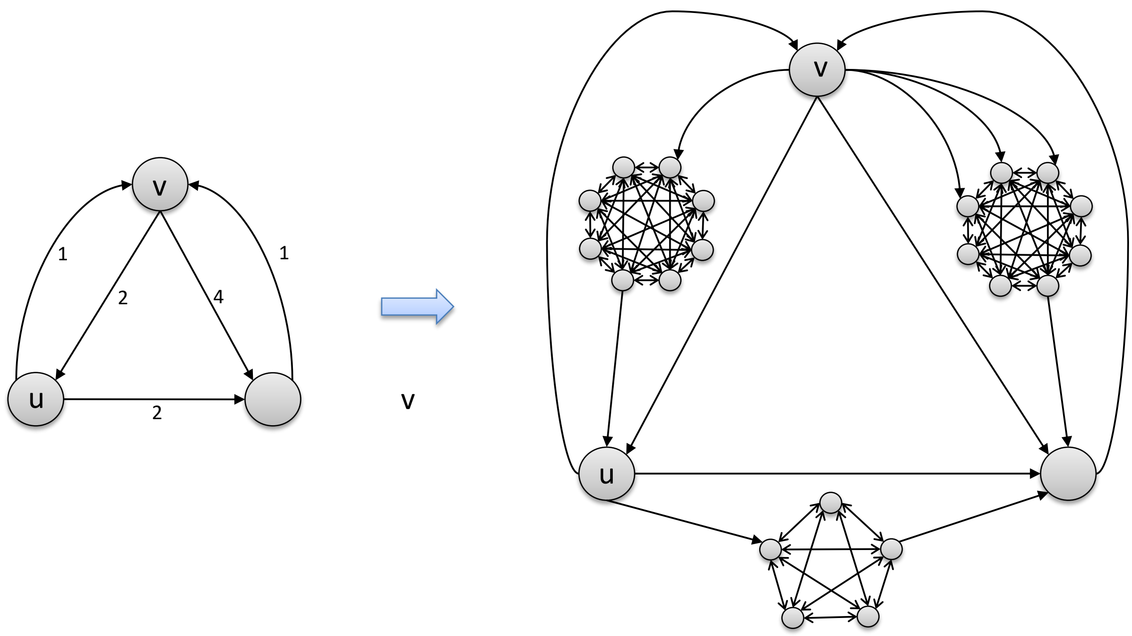

Figure 3.10: Example of an instance G of KFAD and the correspondent gadget in of an instance G of PKFAD.

Suppose that G has a set S of arcs such that $\sum_{e_i \in \varrho^+(v)} P(e_i) = k$ and $G[E(G) \setminus S]$ is knot-free. We build a set S' such that $G'[E'(G') \setminus S']$ is knot-free. For each arc $e_i = (v, u)$ in S we put in S' the additional $P(e_i)$ arcs ($P(e_i) - 1$ arcs from v_i to Q_{e_i} and one from v to u). Since $G[V(G) \setminus S]$ is knot-free, and all additional out-arcs from v created to each deleted e_i in G is also in S', $G'[V(G') \setminus S']$ is knot-free and S' has size exactly k.

Conversely, Suppose that G' has a set of arcs S' with size k and $G'[E(G') \setminus S']$ is knot-free. We create a set S such that $\sum_{e_i \in \varrho^+(v)} P(e_i) \leq k$ and $G'[E'(G') \setminus S']$ is knot-free as follows. Let $G = V(G') \setminus H$, where H is the set of all vertices in all the Q_{e_i} directed complete subgraphs. We make $S = (S' \setminus E(G'[H]))$. If there are arcs of $G'[H]$ in S' in such a way that there is a sink w in $G[V(H) \setminus S']$: by construction, w is a vertex of a directed complete subdigraph Q_{e_i} of size $P(e_i)$ such that $e_i = (v, u)$; we can safely exchange $\varrho^+(w)$ by $\varrho^+(v)$ in S' which turns v into a sink with fewer arc deletions than w (w needs at least one more arc removal than v). Since Q_{e_i} has a path to u and either u had a path to a sink (other than w) that still remain intact or u had a path to w; in this case, since all paths to w are through v, after the exchange, u is now released by v. If there are no arcs of $G'[H]$ in S': since $G'[V(G') \setminus S']$ is knot free, $G[V(G) \setminus S]$ is also knot-free and for each sink v in $G[V(G) \setminus S]$ there are $\sum_{e_i \in \varrho^+(v)} P(e_i)$ arcs in S'. $\square$

Lemma 3.4.3 shows that WKFAD can be solved in polynomial time; however, KFAD can be solved in linear time and through the presented reduction the linearity time is not achieved. We show next that by generalizing the arguments in Lemma 3.2.1, PKFAD can be solved in linear time.

Corollary 3.4.4. *Let G be an instance of* WKFAD.

(a) Let Q be a knot of G. The minimum number of arc deletions in Q to make it knot-free is $\min_{v \in Q}(\sum_{e_i \in \varrho^+(v)} P(e_i))$.

(b) Let $\mathcal{Q} = \{Q_1, Q_2, ..., Q_p\}$ be the non-empty set of all the existing knots in G. The minimum number of arc deletions in G to make it knot-free is $\sum_{i=1}^{p} \min_{v \in Q_i}(\sum_{e_i \in \varrho^+(v)} P(e_i))$.

Proof. The proof follows directly from Lemmas 3.2.1. $\square$

Note that all the knots of a digraph can be identified in linear time as follows: first, find all the SCCs in linear time by running a depth-first search (Cormen et al. [32], pages 615-621); next, as a consequence of Corollary 3.4.4, we can obtain in linear time a set of arcs with minimum cost whose removal turns a given directed graph G into a knot-free directed graph.

Corollary 3.4.5. WKFAD *can be solved in linear time.*

Next is presented wait-for graphs with weighted vertices, i.e, a distributed computation where the processes have priorities. We call WEIGHTED KNOT FREE VERTEX DELETION (WKFVD) the weighted version of KFVD. We show next that PKFVD is reducible to KFVD, thus solvable in polynomial time.

Lemma 3.4.6. *Let G be an instance of* WKFVD. *We can transform G into an instance G' of* KFVD *in polynomial time if for any vertex $v \in V(G)$, $P(v) \leq poly(n)$.*

Proof. We show that WKFVD $\propto$ KFVD. We build G' from G as follows. Set $G' = G$. For each vertex $v_i \in V(G)$ define $P(v_i) - 1$ cycles of size two $(\{U_{v_i^1}, W_{v_i^1}\}, \ldots, \{U_{v_i^{p-1}}, W_{v_i^{p-1}}\})$ and one arc from each vertex U of the directed cycles of size two to the copy of v_i in G' (see Figure 3.11).

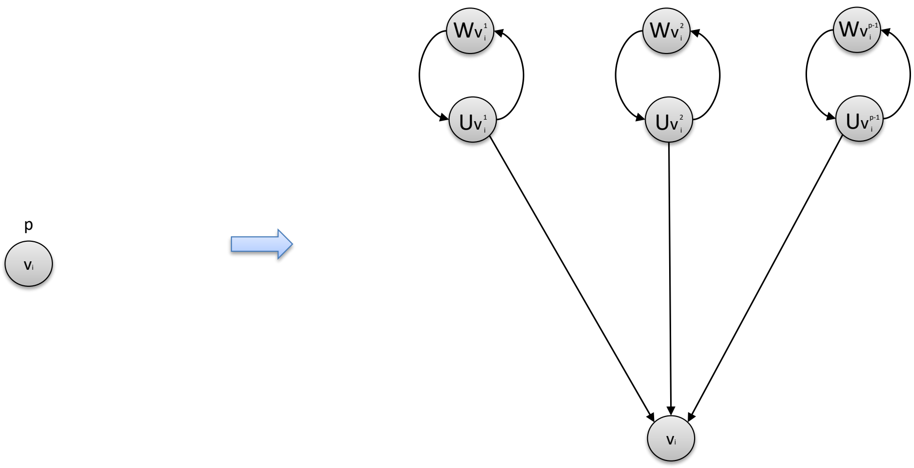

Figure 3.11: Example of vertex v with weight p of an instance G of KFVD and the correspondent gadget in of an instance G of WKFVD.

Suppose that G has a set S such that $\sum_{v_i \in S} P(v_i) = k$ and $G[V(G) \setminus S]$ is knot-free. We create a set S' of size k such that $G'[V(G') \setminus S']$ is knot-free. First set $S' = S$. For each vertex v_i in S', insert $\{U_{v_i^1}, \ldots, U_{v_i^1}\}$ in S'. Since in $G'[V(G') \setminus S]$ the only remaining knots are the additional cycles of size two that had its exit deleted, adding the U vertices corresponding to the vertices in S clearly makes $G'[V(G') \setminus S']$ knot-free.

Conversely, suppose that G' has a set S' of size k such that $G'[V(G') \setminus S']$ is knot-free. We create a set S such that $\sum_{v_i \in S} P(v_i) = k$ such that $G[V(G) \setminus S]$ is knot-free. Let H be the set of vertices $\{U_{v_i^1}, W_{v_i^1}\}$, $\ldots, \{U_{v_i^{p-1}}, W_{v_i^{p-1}}\}$ such that $G = G'[V(G') \setminus H]$. Since $G'[V(G') \setminus S']$ is knot-free and by construction $V(G') = V(G) \cup H$ and no vertex in $V(G) \setminus H$ reaches a vertex in H, $G[V(G) \setminus S']$ is also knot-free. Furthermore, for each deleted vertex $v_i \in (S' \setminus H)$, there are $P(v_i) - 1$ directed cycles of size two; since $G'[V(G') \setminus S']$ is knot-free, at least one vertex in each of these cycles must be in S'. Finally, by setting $S = (S' \setminus H)$, we have a set S such that $\sum_{v_i \in S} P(v_i) = k$ and $G[V(G) \setminus S]$ is knot-free. $\square$

3.5 Conclusions

We show that Vertex–Deletion(AND) and Arc–Deletion(AND) are equivalent to Directed Feedback Vertex Set and Directed Feedback Arc Set, respectively. We proved that Arc–Deletion(OR) and Output–Deletion(OR) are solvable in polynomial time. In addition, Vertex–Deletion(OR) was shown to be NP-complete. Such results are summarized in Table 3.2.

λ–Deletion($\mathbb{M}$)

$\lambda \setminus \mathbb{M}$	AND	OR	AND-OR	X-Out-Of-Y
Arc	NP-H	P	NP-H	NP-H
Vertex	NP-H	NP-H	NP-H	NP-H

Table 3.2: Computational complexity of λ–Deletion($\mathbb{M}$).

A study of the complexity of Vertex–Deletion(OR) in different graph classes was also done. We proved that the problem remains NP-hard even for strongly connected graphs and planar bipartite graphs with maximum degree $\Delta(G) = 4$. Furthermore, we proved that for graphs with maximum degree three the problem can be solved in polynomial time (see Table 3.3).

Vertex–Deletion(OR)

Instance	Complexity
Weakly connected	NP-Hard
Strongly connected	NP-Hard
Planar, bipartite, $\Delta(G) \geq 4$ and $\Delta(G)^+ = 2$	NP-Hard
$\Delta(G) = 3$	Polynomial
$\Delta(G) = 2$	Trivial
$\Delta(G)^+ = 1$	Trivial

Table 3.3: Complexity of Vertex–Deletion(OR) for some graph classes.

In addition, we explored weighted wait-for graphs, where we show that W–λ–Deletion(OR) can be reduced into λ–Deletion(OR) and W–Arc–Deletion(OR) can also be solved in linnear time. Also, W–λ–Deletion(AND) can be reduced into λ–Deletion(AND).

Chapter 4

Fine-Grained Parameterized Complexity Analysis

In this chapter we present a fine-grained parameterized analysis of KFVD. First we show that the KFVD problem is W[1]-hard when parameterized by the size of the input. Then, we show that: KFVD can be solved in $2^{k \log \varphi} n^{O(1)}$ time, but assuming SETH it cannot be solved in $(2-\epsilon)^{k \log \varphi} n^{O(1)}$ time, where φ is the size of a largest strongly connected subgraph of G; KFVD can be solved in $2^{\phi} n^{O(1)}$ time, but assuming ETH it cannot be solved in $2^{o(\phi)} n^{O(1)}$ time, where ϕ is the number of vertices with out-degree at most k; unless $NP \subseteq coNP/poly$, KFVD does not admit polynomial kernel even when $\varphi = 2$ and k is the parameter; KFVD can be solved in time $2^{O(tw)} \times n$, but assuming ETH it cannot be solved in $2^{o(tw)} \times n^{O(1)}$, where tw is the treewidth of the underlying undirected graph

4.1 On the solution size as parameter

In this section, we show that unless $FPT = W[1]$, there is no FPT-algorithm for k–Knot-Free Vertex Deletion. The Knot-Free Vertex Deletion problem parameterized by the size of the solution is formally presented next.

k–Knot-Free Vertex Deletion (k–KFVD)
Instance: A directed graph $G = (V, E)$ and a positive integer k.
Parameter: k (The size of solution).
Question: Determine if G has a set $S \subset V(G)$ such that $\lvert V \rvert \leq k$ and $G[V \setminus S]$ is knot-free.

We present a simple and useful reduction for the reader to get more familiar with the problem. Later we present modifications to it in order to obtain results regarding some width parameterizations.

Theorem 4.1.1. *The k-KFVD problem is W[1]-hard.*

Proof. The proof is based on an FPT-reduction from k-Multicolored Independent Set (k-MIS), a well-known W[1]-complete problem [29]. Let (G', k') be an instance of Multicolored Independent Set, and let $V^1, V^2, \ldots, V^{k'}$ be the color classes of G'. We construct an instance (G, k) of Knot-Free Vertex Deletion as follows (see Fig. 4.1):

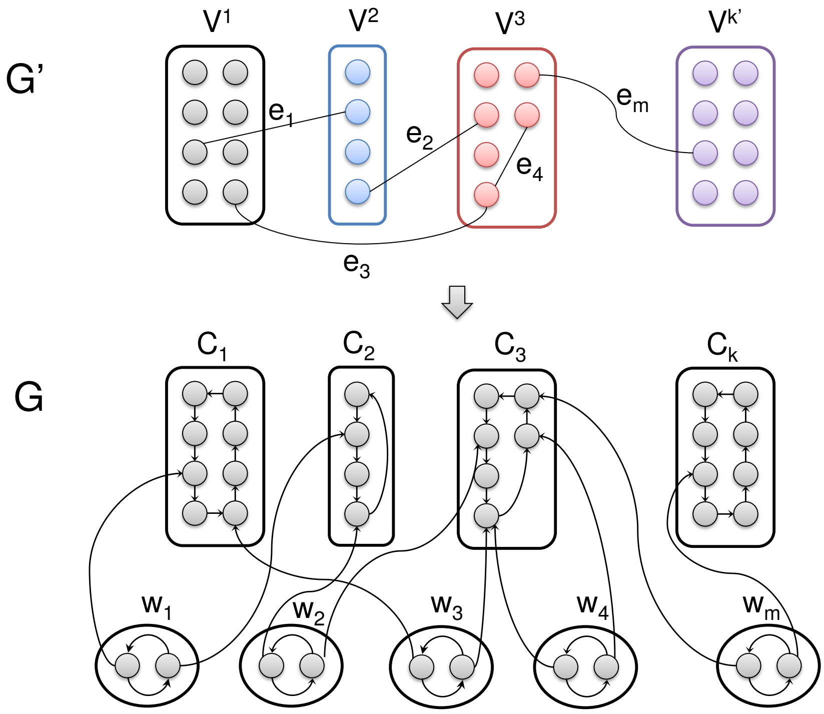

Figure 4.1: Instance (G', k') of Multicolored Independent Set and instance (G, k) of k–KFVD.

1. for each vertex v' in G', create a vertex v in G;
2. for a color class V^i in G', create a directed cycle C_i with its corresponding vertices in G;
3. for each edge $e_j = (u', v')$ in G' create a strongly connected component (scc) W_j with two artificial vertices, u_j^w and v_j^w;
4. for each artificial vertex v_j^w, create an edge from v_j^w towards v in G;
5. finally, set $k = k'$.

Suppose that S' is a k-independent set with exactly one vertex of each set V^i of G'. By construction, G has k knots, one for each color class V^i in G'. Thus, at least k vertex removals are necessary to make G knot-free. We set $S = \{v \mid v' \in S'\}$. Next, we show that $G[V \setminus S]$ is knot-free. Each knot C_i is an induced cycle of G, and it is associated with a color class V^i of G'. Since S' has one vertex of each color class V^i, all induced cycles C_i will be turned into directed paths after the removal of S. Now, it only remains to show that no new knots appear after the removal of S. Notice that S' is a k-independent set of G'; thus, each SCC W_j in G is adjacent to at least one vertex that is not in S. Hence, each SCC W_j will have at least one of its exits preserved, i.e., no new knots are created.

Conversely, suppose that G has a set of vertices S of size k such that $G[V \setminus S]$ is knot-free. Note that G has k knots. Then, exactly one vertex of each cycle C_i is in S. By deleting S, each cycle C_i related to V^i will be turned into a path, and no new knots are created after the deletion of S; thus, every scc W_j will have at least one of its exits preserved. We set $S' = \{v' \mid v \in S\}$. Since each scc W_j corresponds to an edge of G', and at least one vertex of each edge of G' is not in S' (otherwise $G[V \setminus S]$ is not knot-free), S' has no pair of adjacent vertices; moreover, S' is composed by one vertex of each C_i. Therefore S' is a multicolored independent set of G'. □

Corollary 4.1.2. *Assuming ETH, there is no $f(k) \times n^{o(k)}$ time algorithm for* KFVD*, for any computable function f.*

Proof. It is known that Multicolored Independent Set does not admit a $f(k) \times n^{o(k)}$ time algorithm, unless ETH fails (see [41]). As the parameterized reduction presented in Theorem 4.1.1 has linear parameter dependence, we obtain the tight lower bound for KFVD. □

Next, we present two FPT-algorithms for the KFVD problem. The first algorithm takes into account the size of the largest scc and the size k of the solution as aggregated parameters. The second algorithm uses the number of vertices with maximum out-degree at most k as the parameter.

4.2 The size of the largest strongly connected component as an aggregate parameter

In this section, we consider the size of the largest scc of the input directed graph as an additional parameter. The choice of the size of the largest scc as a parameter is mainly inspired by the reductions presented in [24] that prove the NP-hardness of KFVD (even for restricted graph classes). Such reductions result in graphs with scc's of size at most three, and planar graphs with scc's of size at most six.

The W[1]-hardness of k–KFVD, and the NP-completeness of KFVD in graphs having only scc's of small size motivates the following parameterized problem:

$[k, \varphi]$–Knot-Free Vertex Deletion ($[k, \varphi]$–KFVD)
Instance: A directed graph $G = (V, E)$, and a positive integer k;
Parameter: k and φ (the size of a largest scc of G);
Question: Determine if G has a set $S \subseteq V(G)$ such that $\|S\| \leq k$ and $G[V \setminus S]$ is knot-free.

We first describe a $2^{k \log \varphi} \times n^{O(1)}$ time algorithm for $[k, \varphi]$–KFVD.

Lemma 4.2.1. $[k, \varphi]$-KFVD *can be solved in* $2^{k \log \varphi} \times n^{O(1)}$ *time.*

Proof. We use a bounded search tree algorithm. In each node of the search tree, all possible vertices to be removed from the smallest knot of the current graph are analyzed (their number is bounded by φ). Next, for each possibility, one selected vertex is removed, generating a new branch, where the previous steps will be recursively applied until obtaining a knot-free directed graph or removing exactly k vertices. Since any knot has at most φ vertices, the number of levels is bounded by k, all knots in a directed graph can be found and enumerated in linear time with a depth-first search [32], and the deletion of a vertex cannot increase the size of the largest SCC, the algorithm runs in $2^{k \log \varphi} \times n^{O(1)}$ time. Finally, as any knot of a directed graph must have at least one vertex removed, the algorithm checks all possible sets of size at most k that may produce a solution. Thus, the algorithm is correct. □

Algorithm 1 for $[k, \varphi]$–KFVD is presented next.

Lower bounds based on SETH. Now, we show that $[k, \varphi]$-KFVD cannot be solved in $(2-\epsilon)^{k \log \varphi} \times n^{O(1)}$ time, unless SETH fails. To show this lower bound we present a reduction from CNF-SAT to KFVD.

Theorem 4.2.2. *Assuming SETH, there is no* $(2-\epsilon)^{k \log \varphi} \times |V(G)|^{O(1)}$ *time algorithm for* KFVD *for any* $\epsilon > 0$*, where* φ *is the size of a largest strongly connected subgraph of the input.*

Proof. Let F be an instance of CNF-SAT [56] with n variables and m clauses. From F we build a graph $G_F = (V, E)$ which will contain a set $S \subseteq V(G)$ of size $k = n$ such that

Algorithm 1: KFVD(G, k, φ)

```
if G is knot-free then
    return true;
else
    if k = 0 then
        return false;
    end if
end if
answer := false;
Q ← set of vertices of the smallest knot in G;
foreach v_i ∈ Q do
    if G − v_i is knot-free then
        return true;
    else
        answer := answer ∨ KFVD(G − v_i, k − 1, φ);
    end if
    return answer;
end foreach
```

$G[V \setminus S]$ is knot-free if and only if F is satisfiable. The construction of G_F is described below:

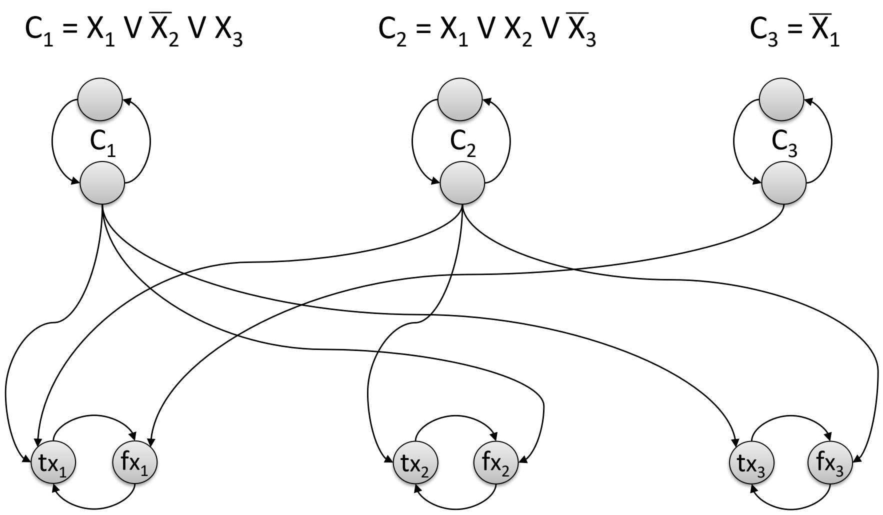

Figure 4.2: The resulting graph $G = (V, E)$ from a formula $F = (x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3) \wedge (\overline{x_1})$ where $|V(G)| = O(n + m)$ and $\varphi = 2$.

1. For each variable x_i in F, create a directed cycle with two vertices ("variable cycle"), t_{x_i} and f_{x_i}, in G_F.

2. For each clause C_j in F create a directed cycle with two vertices ("clause cycle"), $\ell^1_{c_j}$ and $\ell^2_{c_j}$, in G_F.

3. for each literal x_i (resp. $\bar{x_i}$) in a clause C_j, create an arc from $\ell^1_{c_j}$ to t_{x_i} (resp. f_{x_i}).

At this point, it is easy to see that F has a truth assignment if and only if G_F has a set S of vertices containing precisely one vertex of each knot of G_F, such that the removal of S from G_F creates n sinks, for which any clause cycle reaches at least one of them.

Notice that the construction of G_F can be done in polynomial time, $\varphi = 2$ and $k = n$. Therefore, if KFVD can be solved in $(2-\epsilon)^{k \log \varphi} \times |V(G_F)|^{O(1)}$ time for $\epsilon > 0$, then we can solve CNF-Sat in $(2-\epsilon)^n (n+m)^{O(1)}$ time, i.e., SETH fails. □

The reduction described above is more restrictive than a SERF reduction, i.e. it is a polynomial reduction that preserves the parameter n.

Lower bound on the kernelization. Next, we present some lower bounds on the size of a kernel to $[k, \varphi]$-KFVD and k-KFVD. We show that, unless $NP \subseteq coNP/poly$, through a PPT-reduction from the Red-Blue Dominating Set (RBDS) Problem that the KFVD problem does not admit polynomial kernel, even if the input graph has scc's of bounded size.

Theorem 4.2.3. *Unless $NP \subseteq coNP/poly$, k-KFVD does not admit a polynomial kernel, even when a largest scc of the input graph G has size 2.*

Proof. In Red-Blue Dominating Set (RBDS) we are given a bipartite graph $G = (B \cup R, E)$ and an integer k and ask whether there exists a vertex set $R' \subseteq R$ of size at most k such that every vertex in B has at least one neighbor in R'. RBDS parameterized by $(|B|, k)$ is equivalent to Small Universe Set Cover, and RBDS parameterized by $(|R|, k)$ is equivalent to Small Universe Hitting Set. Both problems were shown to have no polynomial kernel (see [45]), unless $NP \subseteq coNP/poly$.

The proof is a PPT-reduction from RBDS parameterized by $(|R|, k)$. Let (G, k) be an instance of RBDS parameterized by $(|R|, k)$. We build an instance (G', k') of Knot-Free Vertex Deletion as follows (see Fig. 4.3):

1. set $k' = |R| + k$;
2. for each vertex v_i in R, create in G' a weakly connected component C_i as follows:
   (a) create two directed cycles of size two, (c^1_i, c^2_i) and (c^3_i, c^4_i);
   (b) create an edge from c^3_i towards c^2_i.

3. for each vertex u_j in B create a set $W_j = \{C_j^1, C_j^2, \ldots, C_j^{k'+1}\}$, were each C_j^z is a directed cycle of size two;

4. finally, for each edge (v_i, u_j) in G, create one directed edge from a vertex of each $C_j^z \in W_j$ to the vertex c_i^1.

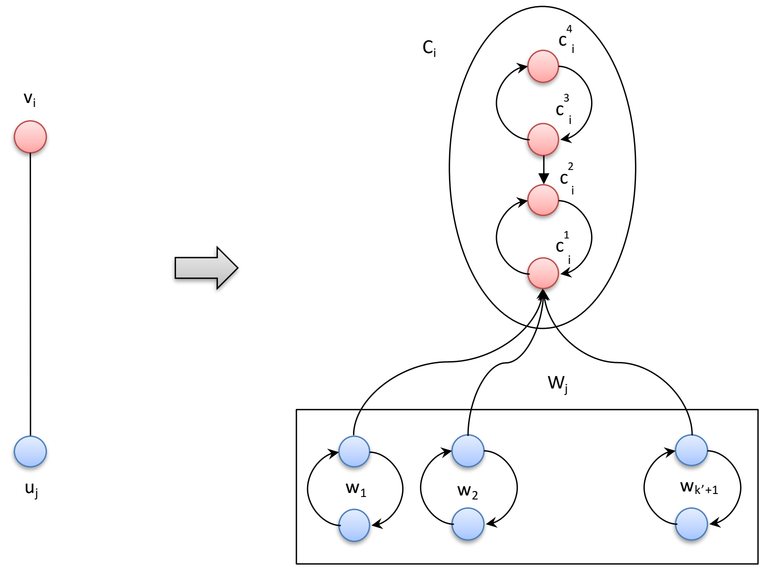

Figure 4.3: PPT-reduction from RBDS parameterized by $(|R|, k)$ to k–KFVD with $\varphi = 2$.

Suppose that S is a red/blue dominating set of G with size k. We build from S a knot-free vertex deletion set S' of G' with size $|R|+k$ as follows: for each vertex $v_i \in R$ we add c_i^1 to S' if $v_i \notin S$, and add c_i^2 and c_i^3 to S' if $v_i \in S$. Since S is a red/blue dominating set of G, every cycle in each W_j will have an arc pointing to one sink c_i^1 in $G'[V \setminus S']$. In addition, all other vertices have either turned into sinks or reach a sink in $G'[V \setminus S']$. Therefore $G'[V \setminus S']$ is knot-free, and $|S'| = |R| + k$.

Conversely, suppose that G' has a set S' of size $k' = |R| + k$ such that $G'[V \setminus S']$ is knot-free. We build from S' a red/blue dominating set S of G with size at most k as follows: add vertex v_i in S if $c_i^1 \notin S'$. Now, we show that S is a red/blue dominating set of G. First observe that G' has $|R|$ knots, and for each $v_i \in R$, $\{c_i^1, c_i^2\}$ induces a knot of G'; then either $c_i^1 \in S'$ or $c_i^2 \in S'$. In addition, for any $v_i \in R$ the removal of c_i^2 creates another knot induced by $\{c_i^3, c_i^4\}$; thus $c_i^1 \notin S'$ implies $c_i^2 \in S'$, and hence $\{c_i^3, c_i^4\} \cap S' \neq \emptyset$. Since G' has $|R|$ knots and $|S'| = |R| + k$, it follows that $|S| \leq k$. Also, since each W_j has $|R| + k + 1$ cycles, without loss of generality we can assume that no vertex in W_j is in S', and as $G'[V \setminus S']$ is knot-free, any vertex in W_j (representing a blue vertex) reaches a sink in $G'[V \setminus S']$, which by construction is a vertex c_i^1 (representing a red vertex). Then S is a set of red vertices with size at most k that dominates all blue vertices of G. $\square$

Corollary 4.2.4. $[k, \varphi]$-KFVD *does not admit a kernel of size* $k^{f(\varphi)}$*, unless* $NP \subseteq coNP/poly$.

Proof. This follows from Theorem 4.2.3 and the fact that a kernel of size $k^{f(\varphi)}$ for $[k, \varphi]$-KFVD would be a polynomial kernel for k-KFVD when a largest scc of the input graph G has size 2. □

4.2.1 The number of vertices with few out-edges as parameter

An interesting property related to the degree of the vertices is that if we are interested in removing a set S with k vertices to obtain a knot-free directed graph, then the out-neighbors of the vertices that will be turned into sinks are contained in S. Thus, if we look for only k removals to obtain a knot-free directed graph, then the candidate vertices to become sinks are the vertices with out-degree at most k. At this point, we consider the number of vertices with out-degree at most k as a parameter.

ϕ–Knot-Free Vertex Deletion (ϕ–KFVD)
Instance: A directed graph $G = (V, E)$, and a positive integer k;
Parameter: ϕ (the number of vertices $v \in G$ with $deg^+(v) \leq k$);
Question: Determine if G has a set $S \subseteq V(G)$ such that $\vert S\vert \leq k$ and $G[V \setminus S]$ is knot-free.

Theorem 4.2.5. ϕ–KFVD *can be solved in* $2^{\phi} \times n^{O(1)}$ *time. In addition, it cannot be solved in* $2^{o(\phi)} \times n^{O(1)}$ *time, unless ETH fails.*

Proof. Let L be a set of vertices with $deg^+(v_i) \leq k$ of an input graph G. By Corollary 5.1.4 to solve ϕ–KFVD in $2^{\phi} \times n^{O(1)}$ time, it is only needed to try the deletion of all out-neighbors of the subsets of L, checking if the deletion does not exceed k vertices and if the resulting directed graph is knot-free.

In order to show a lower bound based on ETH to ϕ–KFVD, we can transform an instance F of 3-CNF-SAT to an instance G_F of KFVD using the reduction presented in Theorem 4.2.2, obtaining in polynomial time a graph with $|V| = O(n + m)$, $|E| = O(n + m)$, and $\phi = O(n + m)$. □

Algorithm 2 for ϕ–KFVD is presented next.

Algorithm 2: KFVD(G, k, ϕ)

$H \leftarrow$ set of vertices $v_i \in V(G)$ such that $\deg^+(v_i) \leq k$;
if $|H| > \phi$ **then**
    return *false*;
end if
foreach *subset* $H' \subseteq H$ **do**
    if $\sum\limits_{v_j \in H'} deg^+(v_j) \leq k$ **then**
        $S \leftarrow \bigcup\limits_{v_j \in H'} N^+(v_j)$;
        if $G[V(G) \setminus S]$ *is knot-free* **then**
            return *true*;
        end if
    end if
end foreach
return *false*;

We show through a PPT-reduction from the RBDS that, unless $NP \subseteq coNP/poly$, KFVD does not admit polynomial kernel, even if the input graph has scc's of bounded size. The reduction is a slight modification of Theorem 4.2.3.

Corollary 4.2.6. *Unless $NP \subseteq coNP/poly$,* KFVD *does not admit polynomial kernel when parameterized by k and ϕ.*

Proof. We start by making the same transformation as in Theorem 4.2.3, obtaining a directed graph G. Now, for each scc associated with a blue vertex, we add k auxiliary vertices and add edges in order to transform the component into a complete directed subdigraph with $k+2$ vertices and $(k+2)(k+1)$ arcs. Now, the resulting graph G has $\phi = 4|R|$, and the rest of the proof follows as in Theorem 4.2.3. $\square$

4.3 Conclusions

In this chapter, we study the Knot-Free Vertex Deletion problem from a parameterized complexity point of view. We proved that KFVD with the natural parameter k is W[1]-hard through a FPT-reduction from Multicolored Independent Set, a well-known W[1]-complete problem [29]. Next, we propose two FPT-algorithms, each exploring a different additional parameter. The first parameter, φ, is the maximum size of a SCC of the input graph. We show that KFVD can be solved in $2^{k \log \varphi} n^{O(1)}$ time and unless SETH fails it cannot be solved in $(2-\epsilon)^{(k \log \varphi)} n^{O(1)}$ time. We show that, Unless

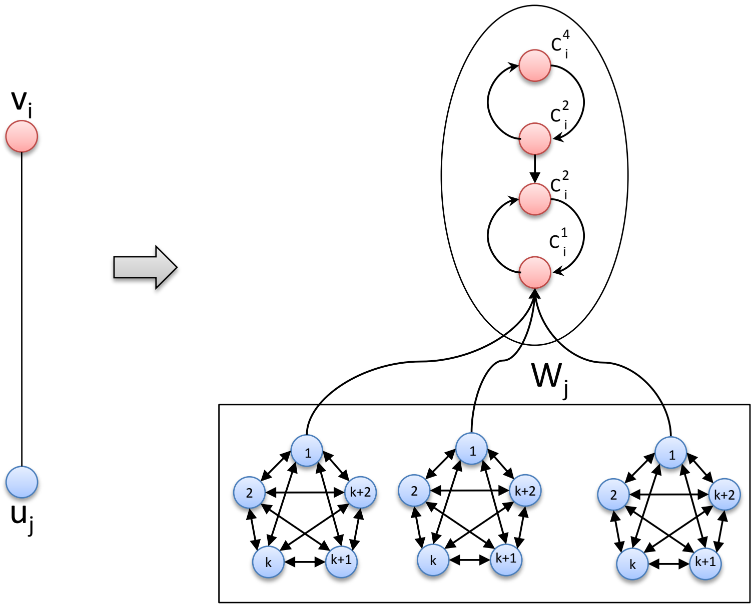

Figure 4.4: PPT-reduction from RBDS parameterized by $(|R|, k)$ to ϕ–KFVD with $\phi = 4|R|$.

$NP \subseteq coNP/poly$, k-KFVD has no polynomial kernel even if the input graph has only SCCs with size bounded by 2. The second algorithm runs in $2^{\phi}n^{O(1)}$ time and it is appropriate for graphs where there are few vertices, ϕ, with small out-degree. In addition, assuming ETH, we show that it cannot be done in $2^{o(\phi)n^{O}(1)}$ time. We also show that, unless $NP \subseteq coNP/poly$, KFVD has no polynomial kernel when the number of vertices with out-degree at most k is a parameter.

Table 4.1 summarizes the results presented in this work.

Table 4.1: Fine-grained parameterized complexity of KNOT-FREE VERTEX DELETION.

		Complexity	Running time	Lower bounds assuming (S)ETH
Parameter	k	W[1]-hard	n^k	no $f(k) \times n^{o(k)}$ alg.
	k, φ	FPT	$2^{k \log \varphi} \times n^{O(1)}$	no $(2-\epsilon)^{k \log \varphi} \times n^{O(1)}$ alg.
	ϕ	FPT	$2^{\phi} \times n^{O(1)}$	no $2^{o(\phi)} \times n^{O(1)}$ alg.

Chapter 5

Width Parameterizations

In this chapter we present a parameterized complexity study of KFVD on directed width measures. First we show that the KFVD problem is W[1]-hard when parameterized by the size of the solution even if the input graph has a K-width 2 and largest directed path of size 5. Then, we propose two FPT-algorithms, each exploring a different additional parameter to the *directed feedback vertex set number* (dfv). The first, combining with K-width (κ), it can be solved in $2^{O(\kappa dfv^5)}n^{O(1)}$. The second, combining with the length of the longest directed path p, it can be solved in $2^{O(dfv^3)}\ p^{O(dfv)}n^{O(1)}$. Also, a $2^{|F|} \times n^c$ time algorithm is presented when we are given a special directed feedback vertex set F whose removal returns an acyclic graph having path cover bounded by a constant c. We show that: KFVD can be solved in FPT time when parameterized by cliquewidth of the underlying undirected graph. Finally, KFVD can be solved in time $2^{O(tw \log tw)} \times n$, but assuming ETH it cannot be solved in $2^{o(tw)} \times n^{O(1)}$, where tw is the treewidth of the underlying undirected graph.

5.1 Preliminaries

In this section, we present some useful remarks and reduction rules. Remind that in the decision version of the problem we are given G and a positive integer k.

The first observation is immediate, as if we can make the graph acyclic, then it will be knot-free.

Observation 5.1.1. *If $k \geq dfv(G)$ then G is a* yes*-instance.*

The two other observations are less obvious but rather natural.

Observation 5.1.2. *Let S be a solution with set of sinks Z in $G_{\bar{S}}$, and $s \in S$. Let $S' = S \setminus \{s\}$ and Z' be the set of sinks of $G_{\bar{S'}}$. If there is a path from s to Z' in $G_{\bar{S'}}$ then S' is also a solution.*

Proof. Let $u \in V(G_{\bar{S'}})$. Let us prove that u has a path to Z' in $G_{\bar{S'}}$. If $u = s$ then it is clear by assumption. Suppose now that $u \neq s$. As S is a solution, let P be a uz-path in $G_{\bar{S}}$ from u to a sink $z \in Z$. As $V(G_{\bar{S}}) \subseteq V(G_{\bar{S'}})$, P still exists in $G_{\bar{S'}}$. Thus, if $z \in Z'$ we are done. Otherwise, it implies that there is $s \in N^+(z)$ such that $P' = (u, \ldots, z, s)$ is a us-path in $G_{\bar{S'}}$. As s has a path to Z' in $G_{\bar{S'}}$, we obtain the desired result. □

Informally, after deleting a vertex s, we can add s back to the graph when it is certain that s has a path to a sink in the current graph. This is detailed by the following lemma and its corollary.

Lemma 5.1.3. *Let S be a solution with set of sinks Z in $G_{\bar{S}}$. If there exists $s \in S$ with $s \notin N^+(Z)$, then $S' = S \setminus \{s\}$ is also a solution.*

Proof. Let Z' be the set of sinks of $G_{\bar{S'}}$. According to Observation 5.1.2, it suffices to prove that there is a path from s to Z' in $G_{\bar{S'}}$. If s is a sink in $G_{\bar{S'}}$ we are done. Otherwise, there exists an arc su in $G_{\bar{S'}}$, with $u \in V(G_S)$. As S is a solution, either u is a sink and we are done, or, there exists a uz-path P in $G_{\bar{S}}$ with $z \in Z$. As $V(G_{\bar{S}}) \subseteq V(G_{\bar{S'}})$, P still exists in $G_{\bar{S'}}$, and $s \notin N^+(Z)$, z is still a sink in $G_{\bar{S'}}$. □

The following corollary is immediate.

Corollary 5.1.4. *In any optimal solution S with set of sinks Z in $G_{\bar{S}}$, we have $N^+(Z) = S$.*

Observation 5.1.5. *Let S be a knot-free vertex deletion with set of sinks Z in $G_{\bar{S}}$. If $|S| \leq k$ then for any vertex v with $d^+(v) > k$ it holds that $v \notin Z$.*

To complete the previous observations, we present two general reduction rules.

Reduction rule 5.1.6. *If $v \in V(G)$ is an SCC of size one then remove $A[v]$.*

Proof. Let G' be the graph obtained by removing $A[v]$. Let of first show that (G, k) is a *yes*-instance implies that (G', k) is also a *yes*-instance. Let S be a solution of G of size at most k with set of sinks Z in $G_{\bar{S}}$. Let $S' = S \setminus A[v]$, and Z' the set of sinks in $G'_{\bar{S'}}$. Let us prove that every $u \in V(G'_{\bar{S'}})$ has a path ot Z' in $G'_{\bar{S'}}$. Let $u \in V(G'_{\bar{S'}})$. As u is also in $V(G_{\bar{S}})$, there is a uz-path P in $G_{\bar{S}}$ where $z \in Z$. As $u \notin A[v]$, $V(P) \cap A[v] = \emptyset$ and thus, the path P still exists in $G'_{\bar{S}'}$. Moreover, $u \notin A[v]$ implies that $N^+(z) \cap A[v] = \emptyset$, and thus that $N^+(v) \subseteq S'$, implying that $z \in Z'$.

Let us now consider the reverse implication, and let S' be a solution of G' of size at most k with set of sinks Z' in $G'_{\bar{S}'}$ and prove that S' is a solution of G. Let us start with $u \in V(G_{\bar{S}'}) \setminus A[v]$. As S' is a solution of G' and $u \in V(G'_{\bar{S}'})$, there is uz'-path P' in $G'_{\bar{S}'}$ where $z' \in Z'$, and this path still exists in $G_{\bar{S}'}$. As $N^+(z') \cap A[v] = \emptyset$, z' is still a sink in $G_{\bar{S}'}$ and we are done. Consider now a vertex $u \in V(G_{\bar{S}'}) \cap A[v]$. As $S' \cap A[v] = \emptyset$, there is uv-path P in $G_{\bar{S}'}$. If $N^+(v) \subseteq S'$ then v is a sink in $G_{\bar{S}'}$ and we are done. Otherwise, let $w \in N^+(v) \setminus S'$. As v is an SCC of size 1, $N^+(v) \cap A[v] = \emptyset$, implying that $w \in V(G_{\bar{S}'}) \setminus A[v]$, and thus according to the previous case w has a path to a sink in $G_{\bar{S}'}$. $\square$

The previous reduction rule removes in particular sources and sinks, as they are SCC's of size one.

Reduction rule 5.1.7. *Let U_i be a strongly connected component of G with strictly more than k out-neighbors in $G[V \setminus V(U_i)]$. Then we can safely remove $A[U_i]$.*

Proof. Let G' be the graph obtained by removing $A[U_i]$. Let us first show that (G, k) is a *yes*-instance implies that (G', k) is also a *yes*-instance. Let S be a solution of G of size at most k and Z the set of sinks in $G_{\bar{S}}$. Let $S' = S \setminus A[U_i]$, and Z' the set of sinks in $G'_{\bar{S}'}$. Using the same argument (replacing $A[v]$ by $A[U_i]$) as in the first part of proof of Reduction 5.1.6, we get that every $u \in V(G'_{\bar{S}'})$ has a path ot Z' in $G'_{\bar{S}'}$.

Let us now consider the reverse implication, and let S' be a solution of G' of size at most k with set of sinks Z' in $G'_{\bar{S}'}$ and prove that S' is a solution of G. Let us start with $u \in V(G_{\bar{S}'}) \setminus A[v]$. As S' is a solution of G' there is uz'-path P' in $G'_{\bar{S}'}$ where $z' \in Z'$, and this path still exists in $G_{\bar{S}'}$. As $N^+(z') \cap A[U_i] = \emptyset$, z' is still a sink in $G_{\bar{S}'}$ and we are done. Consider now a vertex $u \in V(G_{\bar{S}'}) \cap A[U_i]$. As $S' \cap A[U_i] = \emptyset$, there is uU_i-path P in $G_{\bar{S}'}$. As U_i has strictly more than k out-neighbors in $G[V \setminus V(U_i)]$, there is arc from U_i to $w \in V(G_{\bar{S}'})$ and thus according to the previous case w has a path to a sink in $G_{\bar{S}'}$. $\square$

At this point, we may assume that reduction rules 5.1.6 and 5.1.7 do not apply to the input digraphs G.

5.2 Directed width measures

In Theorem 4.1.1, k-KFVD was shown to be W[1]-hard using a reduction from k-Multicolored Independent Set (k-MIS). However, the gadget used in this reduction to encode each color class has the longest directed path of unbounded length. First, we remark that it is possible to modify the reduction in order to prove that k-KFVD is W[1]-hard even if the input graph G has bounded longest path length and K-width.

Theorem 5.2.1. k-KFVD *is W[1]-hard even if the input graph has longest directed path of length at most* 5 *and K-width equal to* 2.

Proof. Let (G', k) be an instance of k-MIS, and let $V^1, V^2, \ldots, V^k$ be the color classes of G'. We construct an instance (G, k) of Knot-Free Vertex Deletion with bounded longest path length and K-width as follows. (see Figures 5.1 and 5.2):

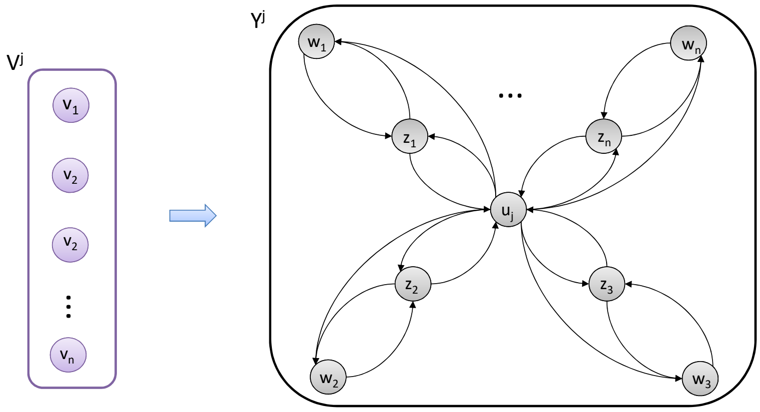

Figure 5.1: Vertex gadget Y_j in G built from the set of vertices of color V^j in G'.

1. for each $v_i \in V(G')$, create a directed cycle of size two with the vertices w_i and z_i in G;

2. for a color class V^j in G', create one vertex u_j;

3. for each vertex z_i in G corresponding to a vertex v_i of the color class V^j in G', create an arc from z_i to u_j and from u_j to z_i.

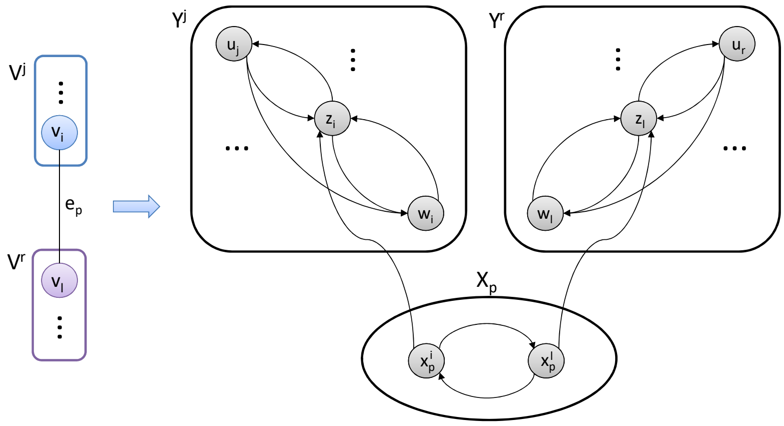

Figure 5.2: An arc cycle X_p built from a arc $e_p = (v_i, v_l)$ in G' and its connection to appropriated vertices in the vertex gadgets.

4. for each vertex w_i in G corresponding to a vertex v_i of the color class V^j in G', create an arc from u_j to w_i
5. for each edge $e_p = (v_i, v_l)$ in G' create a set X_p with two artificial vertices x_p^i and x_p^l and the arcs $x_p^i x_p^l$ and $x_p^l x_p^i$;
6. for each artificial vertex x_p^i, create an edge from x_p^i towards z_i in G.

Finally, set $Y_j = \{w_i, z_i \ : \ v_i \in V^j\} \cup \{u_j\}$, Y_j is the set of vertices of G corresponding to the vertices of G' in the same color class V^j. Notice that, the longest path of G has at most 5 vertices, and for any pair s, t in $V(G)$ there are at most 2 distinct directed st-paths in G. The rest of the proof is similar to Theorem 4.1.1.

□

After the introduction of the notion of directed treewidth (dtw) [65], a large number of width measures in digraphs were developed, such as: cycle rank [57] (cr); directed pathwidth [5] (dpw); zig-zag number [76] (zn); Tree-Zig-Zag number [77] (Tzn); Kelly-width [61] (Kelw); DAG-width [12] (dagw); D-width [84] (Dw); weak separator number [77] (s); entanglement [13] (ent); DAG-depth [54] (ddp). However, if a graph problem is hard when both the longest directed path length and the K-width are bounded, then it is hard for all these measures (see Figure 5.3).

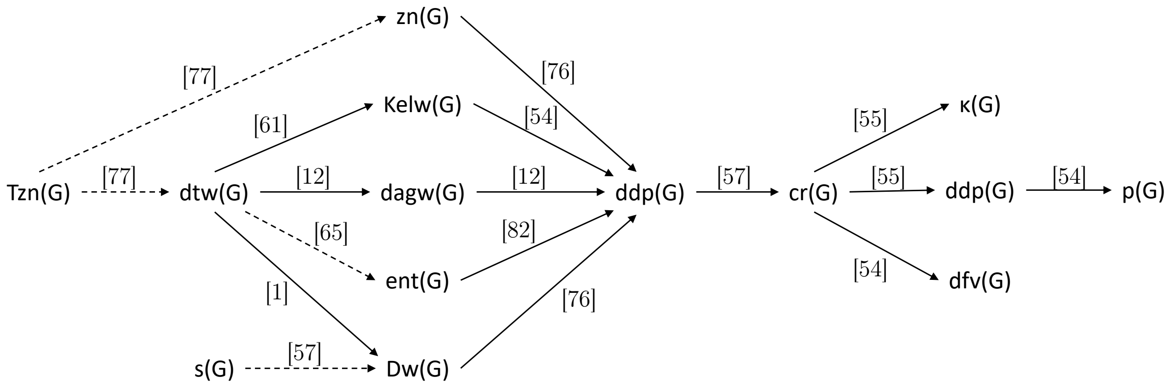

Figure 5.3: A hierarchy of digraph width measure parameters. $\alpha \rightarrow \beta$ indicates that $\alpha(G) \leq f(\beta(G))$ for any digraph G and some function f. More details about the relationships between these parameters can be found in the references corresponding to each arrow.

Therefore, from the reduction presented in Theorem 5.2.1 we can observe that KFVD is para-NP-hard concerning all these width measures, and k–KFVD is W[1]-hard even on inputs where such width measures are bounded.

Thus, it seems to be extremely hard to identify helpful width parameters for which KFVD can be solved in FPT-time or even in XP-time. Fortunately, there remain some parameters for which, at least, XP-time solvability is achieved. One of them is the *directed feedback vertex set number* (dfv). This invariant is an upper bound on the size of a minimum knot-free vertex deletion set, so XP-time algorithms are trivial. This parameter is discussed in more detail in Section 5.3. Another interesting width parameter for directed graphs G that is not bounded by a function of the K-width and the length of a longest directed path is the clique-width of G, which we will discuss in Section 5.4.

5.3 On the size of a minimum directed feedback vertex set as parameter

Recall that k–KFVD is $W[1]$-hard (for fixed K-width and longest directed path) and that, as noticed in Observation 5.1.1, we can assume $k < dfv(G)$, this motivates us to determine the status of dfv–KFVD. First, we present two FPT-algorithms, both with the size of a minimum directed feedback vertex set as a parameter but with an aggregate parameter, the K-width, $\kappa(G)$, for the first one and the length of a longest directed path, $p(G)$, for the second one. Since finding a minimum directed feedback vertex set F in G

can be solved in FPT-time (with respect to dfv) [28], we consider that F, a minimum DFVS, is given. Namely, we show that both (dfv, κ)–KFVD and (dfv, p)–KFVD are FPT.

At this point, we need to define the following variant of KFVD.

Disjoint Knot-Free Vertex Deletion (Disjoint-KFVD)
Instance: A directed graph $G = (V, E)$; a subset $X \subseteq V$; and a positive integer k.
Question: Determine if G has a set $S \subset V(G)$ such that $\|S\| \leq k$, $S \cap X = \emptyset$ and $G[V \setminus S]$ is knot-free.

We call *forbidden vertices* the vertices of the set X. It is clear that Disjoint-KFVD generalizes KFVD by taking $X = \emptyset$.

Let us now define two more steps that are FPT parameterized by dfv, and that will be used for both (dfv, κ)–KFVD and (dfv, p)–KFVD. The next step will allow us to consider that the vertices of F are forbidden. We need the following straightforward observation.

Observation 5.3.1. *Let (G, k) be an instance of* KFVD *and $v \in V(G)$.*

- *if (G, k) is a* yes-*instance and there exists a solution S with $v \in S$, then $(G \setminus \{v\}, k-1)$ is a* yes-*instance*
- *if $(G \setminus \{v\}, k-1)$ is a* yes-*instance then (G, k) is a* yes-*instance*

Branching 5.3.2 (On the directed feedback vertex set $\boldsymbol{F}$)**.** *Let (G, F, k) be an instance of dfv–*KFVD*. In time $2^{dfv} \times O(m)$ we can build 2^{dfv} instances (G^i, F_1^i, X^i, k^i) of dfv-*Disjoint-KFVD *as follows. We consider all possible partitions of F into two parts: F_1, the set of vertices of F that will not be removed (i.e., they become forbidden); and F_2, the set of vertices in F that will be removed. For each such a partition (indicated by the index i), we remove the set F_2^i of vertices and we apply Reduction Rules 5.1.6 and 5.1.7 until they are no longer applicable (see Section 5.1). We denote by G^i the obtained graph, $X^i = F_1^i$, and $k^i = k - |F_2^i|$.*

Using Observation 5.3.1, the following lemma is immediate.

Lemma 5.3.3. *(G, F, k) is a* yes-*instance of dfv–*KFVD *if and only if one of the instances (G^i, F_1^i, X^i, k^i), $1 \leq i \leq 2^{dfv}$, of dfv-*Disjoint-KFVD *is a* yes-*instance.*

Since there are at most 2^{dfv} partitions of F, the branching reduction can be performed in FPT time. Although at this point, $X^i = F_1^i$, in the next steps, some vertices of $V(G) \setminus F_1^i$

may become forbidden and therefore, should be added to X^i. From this point forward, we assume that we are given an instance (G, F_1, X, k) of dfv-DISJOINT-KFVD such that $F_1 \subseteq X$.

Notice that after applying Reduction Rule 5.1.6 (Section 5.1), each strongly connected component of G is at least of size two. Thus, each of them must contain at least one cycle; therefore, the number of strongly connected components of G is bounded by dfv. Moreover, for any strongly connected component U of G, Reduction Rule 5.1.7 gives an upper bound for the number of vertices in $N^+(V(U))$ (i.e., vertices that are not in U but it is out-neighbor of some vertex in U), which implies that G has at most $dfv \times k \leq dfv^2$ such vertices between its strongly connected components. This observation leads to a branching rule.

Branching 5.3.4 (On strongly connected components)**.** *Let S_H be the set of vertices that are* heads *of arcs between the strongly connected components of G. We have $|S_H| \leq dfv \times k \leq dfv^2$. Let (G, F, X, k) be an instance of dfv-*DISJOINT-KFVD *with $F \subseteq X$ and such that Reduction Rules 5.1.6 and 5.1.7 do not apply. In time $2^{|S_H|} \times O(m)$ we can build 2^{dfv} instances of dfv-*DISJOINT-KFVD *as follows. We consider all possible partitions (indicated by index i) of S_H into $S_H^{(i,1)}$ (guess of forbideen vertices) and $S_H^{(i,2)}$ (guess of vertices to be deleted) with $S_H^{(i,2)} \cap X = \emptyset$. We add $S_H^{(i,1)}$ to X; remove the set $S_H^{(i,2)}$ (recall Observation 5.3.1); and apply Reduction Rules 5.1.6 and 5.1.7 until they are no longer applicable.*

Notice that this step involves a $2^{|S_H|}$ branching. At this point, we may consider that we have an instance (G, F, X, k) where $F \subseteq X$ and such that for any arc uv between two SCC's U_i and U_j, $v \in X$. We call such an instance as a *nice* instance.

Lemma 5.3.5 (NICE dfv-DISJOINT-KFVD)**.** *If there is an algorithm running in $g(dfv) \times poly(n)$ time for dfv-*DISJOINT-KFVD *restricted to nice instances that are strongly connected, i.e, G is a knot, then there is an FPT algorithm running in $g(dfv) \times poly(n) \times k \times \log(k)$ time to solve dfv-*DISJOINT-KFVD *for any nice instance.*

Proof. Let (G, F, X, k) be a nice instance and S be a solution. Let $\mathcal{U} = \{U_1, \ldots, U_s\}$ be the partition of $V(G)$ where each U_i is an SCC, and let $\mathcal{K} = \{U_i : U_i \text{ is a knot}\}$. Without loss of generality we can assume that $\mathcal{K} = \{U_1, \ldots, U_t\}$ for some $t \leq s$. Let $S_i = S \cap U_i$. Notice that if S is a solution then for any $i \in [t]$, S_i is a solution of $(G[U_i], F \cap U_i, X \cap U_i, |S_i|)$. Moreover, given for each $i \in [t]$ a solution S'_i of $(G[U_i], F \cap U_i, X \cap U_i, |S'_i|)$, such that $\sum_{i=1}^{t} |S'_i| \leq k$, it holds that $S' = \bigcup_{i=1}^{t} S'_i$ will be a solution to (G, F, X, k), because vertices of some $U_j \notin \mathcal{K}$ will still have a path to a set $U_i \in \mathcal{K}$ in $G_{\bar{S}'}$ since any arc between two SCC's has forbidden endpoints. Thus, given a nice instance (G, F, X, k) and an algorithm A for a nice instance restricted to one SCC, for any $U_i \in \mathcal{K}$ we perform a binary search to find the smallest k_i such that $A(G[U_i], F \cap U_i, X \cap U_i, k_i)$ answers *yes*, and we answer *yes* iff $\sum_{i=1}^{t} k_i \leq k$. □

From Lemma 5.3.5, we may assume that we have an instance (G, F, X, k) such that $F \subseteq X$ and G is strongly connected. We call such an instance as a *super nice* instance.

5.3.1 Taking the K-width as aggregate parameter

In this section, we prove that (dfv, κ)-Disjoint-KFVD is FPT when restricted to super nice instances and $F \subseteq X$.

Let $F = \{v_1, \ldots, v_{dfv}\}$. For every pair of integers i, j with $1 \leq i, j \leq dfv$ we define $H_{i,j}$ as the (i, j)-*connectivity set*, that is, the set of vertices which are contained in a directed path from v_i to v_j in the induced subgraph $G[V \setminus (F \setminus \{v_i, v_j\})]$ (if $i = j$ then $H_{i,i}$ is the set of vertices contained in a cycle in $G[V \setminus (F \setminus \{v_i\})]$). Let us define a set B on which we will later branch in a way to ensure connectivity among different connectivity sets. We start with $B = \{\emptyset\}$, and then, for each possible pair of connectivity sets $H_{i,j}$, $H_{i',j'}$ we increase B as follows:

i) add $N^+(H_{i,j} \setminus H_{i',j'}) \cap H_{i',j'}$ to B.

ii) add $N^+(H_{i,j} \cap H_{i',j'}) \cap (H_{i',j'} \setminus H_{i,j})$ to B.

iii) add $N^+(H_{i',j'} \setminus H_{i,j}) \cap H_{i,j}$ to B.

iv) add $N^+(H_{i',j'} \cap H_{i,j}) \cap (H_{i,j} \setminus H_{i',j'})$ to B.

For a given pair of connectivity sets, in each of the items *i*), *ii*), *iii*) and *iv*) the number of added vertices to B is at most κ. For instance,let $y_1, \ldots, y_l$ be the vertices added by item *i*), where each $y_s \in N^+(H_{i,j} \setminus H_{i',j'}) \cap H_{i',j'}$. By definition, there exist vertices $x_1, \ldots, x_l$ of $H_{i,j} \setminus H_{i',j'}$ such that $x_s y_s$ are arcs of G for $s = 1, \ldots, l$. Notice that while the y_s's are distinct, the x_s's are not forced to be so. For any $s \in \{1, \ldots, l\}$, there exists a path P_s in $H_{i',j'}$ from y_s to $v_{j'}$, and such a path does not intersect $H_{i,j} \setminus H_{i',j'}$. In the same way, by finding a path Q_s from v_i to x_s for every $s \in \{1, \ldots, l\}$, we form l distinct paths $Q_s P_s$ from v_i to $v_{j'}$, implying $l \leq \kappa$, the K-width of G. So, as there are

dfv^2 different connectivity sets, at the end of the process, B contains at most $4\kappa \times\ dfv^4$ vertices. Figure 5.4 shows examples of vertices to be added in B regarding the interaction of two different connectivity sets.

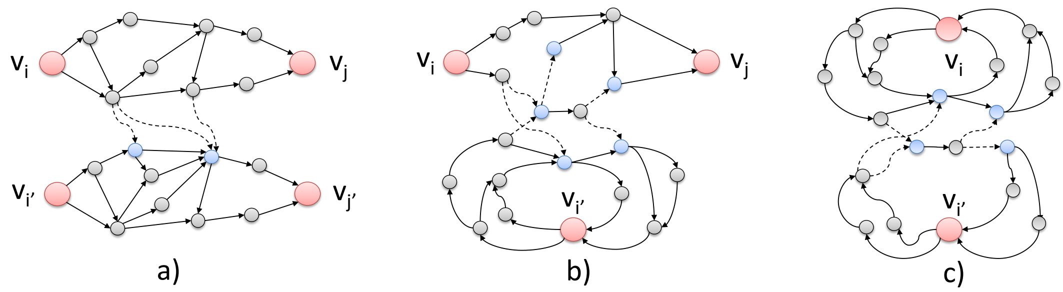

Figure 5.4: a) two connectivity sets with no intersection. b) an intersection with two vertices belonging to both connectivity sets. c) two connectivity sets $H_{i,j}$ with $i = j$. Vertices to be added in B are marked in blue.

Next, we establish our last branching rule.

Branching 5.3.6 (On the connectivity sets)**.** *We branch by partitioning B into two parts: B_1, the set of vertices that will not be removed (ie. they become forbidden); B_2, the set of vertices that will be removed in the branch. Since $|B| \leq 4\kappa \times dfv^4$, we branch at most $2^{4\kappa . dfv^4}$ times.*

At this point, without loss of generality, one can assume that none of the above branching and reductions rules are applicable. Hence, the analysis boils down to the case where all the vertices of $F \cup B$ are forbidden to be deleted ($F \cup B \subseteq X$), and G is strongly connected.

Observation 5.3.7 (The consequences of Branching 5.3.6)**.** *Let G be a graph for which no Reduction Rules 5.1.6 and 5.1.7 or Branchings 5.3.2 to 5.3.6 can be applied. Let $H_{i,j}$ and $H_{i',j'}$ be two different connectivity arc sets in G. If there is an arc from $H_{i,j} \setminus H_{i',j'}$ to $H_{i',j'} \setminus H_{i,j}$ or $H_{i,j} \cap H_{i',j'}$ to $H_{i',j'} \setminus H_{i,j}$ in $G[H_{i,j} \cup H_{i',j'}]$, then the head vertex of such an arc is a forbidden vertex.*

We now aim to show that, for any vertex v^* such that v^* can be turned into a sink, that is, $N^+(v^*) \cap X = \emptyset$, and $d^+(v^*) \leq k$, the deletion of $N^+(v^*)$ is sufficient for G to become knot-free.

Lemma 5.3.8. *Let (G, F, X, k) be an instance of (dfv, κ)-*DISJOINT-KFVD *such that G is strongly connected, $F \subseteq X$, and none of the branching and reduction rules can be applied. If there is a vertex v^* with no forbidden out-neighbor, then $G[V \setminus N^+(v^*)]$ is knot-free.*

Proof. Let (G, F, X, k) and v^* be as stated. Denote by G' the resulting graph, i.e, $G' = G[V \setminus N^+(v^*)]$. For contradiction, assume that G' contains a knot K. As G is strongly connected and K was not a knot in G, this implies that there exists an arc xy of G such that $x \in V(K)$ and $y \in N^+(v^*)$. Notice that $y \notin X$, since y has to be deleted in order to v^* to become a sink. Let us now define a connectivity set containing both y and v^*. Let s be any source of the DAG $G[V \setminus F]$ such that there is a sv^*-path in $G[V \setminus F]$, and let z be any sink of $G[V \setminus F]$ such that there is a yz-path in $G[V \setminus F]$. As G is strongly connected, there exist arcs $v_i s$ and $z v_j$ where $\{v_i, v_j\} \subseteq F$ and we get that $\{v^*, y\} \subseteq H_{i,j}$. Notice that $i = j$ is possible. Similarly, as $G[V(K)]$ is strongly connected, it contains a cycle C' containing x. Thus, there exists a connectivity set $H_{k,l}$ ($k = l$ is possible) such that $\{v_k, v_l\} \subseteq V(C')$, and it contains a path P from v_k to v_l that traverses x and is a subgraph of C'. In addition, v^* is not a vertex of $H_{k,l}$, otherwise there would exist a path P' from v_k to v^* containing no vertex of $F \setminus \{v_k\}$, which is not possible. Indeed, either $V(P') \cap N^+(v^*) = \emptyset$ and we would get that K is not a knot (since v^* is a sink in G'), or $V(P') \cap N^+(v^*) \neq \emptyset$, implying that there is a cycle with no vertex of F. Thus, as y was not a forbidden vertex, it means that $y \notin H_{k,l}$ otherwise the arc v^*y would go from $H_{i,j} \setminus H_{k,l}$ to $H_{i,j} \cap H_{k,l}$ and y should be forbidden by Branching 5.3.6 item i). Then we have $y \in H_{i,j} \setminus H_{k,l}$. Similarly, we have $x \notin H_{i,j} \cap H_{k,l}$, otherwise by item ii) of Branching 5.3.6, vertex y would be forbidden. Finally $x \in H_{k,l} \setminus H_{i,j}$ and $y \in H_{i,j} \setminus H_{k,l}$, since $(H_{i,j} \setminus H_{k,l}) \subseteq H_{i,j}$, and by item iii) of Branching 5.3.6, vertex y would again be a forbidden vertex, a contradiction. $\square$

In conclusion, according to Lemma 5.3.8, we can decide such a super nice instance (G, F, X, k) of (dfv, κ)-Disjoint-KFVD with $F \subseteq X$ as follows. If there exists $v^* \notin F$ with $|N^+(v^*)| \leq k$ and $N^+(v^*) \cap X = \emptyset$ then we answer yes, and otherwise we answer no, i.e, we can find in polynomial time the optimum solution for G by choosing a vertex v^* with minimum out-degree.

Theorem 5.3.9. Knot-free Vertex Deletion *can be solved in* $2^{O(\kappa \,\times\, dfv^5)} \times n^{O(1)}$.

Proof. First, notice that applying Branchings 1 and 2 results in $3^{dfv} \times\ 2^{2dfv^2}$ branches. Branching 3 can be done in time $2^{4\kappa.dfv^4}$, but may re-create several SCC's, forcing us to apply again Branching 2 and reduction rules, but decreasing k; this implies that the total running time of the overall algorithm is $3^{dfv} \times\ (2^{2dfv^2} 2^{4\kappa.dfv^4})^k \times n^{O(1)}$.

Recall that the algorithm generates a set of nice instances (G', F', X', k') of Disjoint-KFVD, such that the input graph G has a knot-free vertex deletion set of size at most k if and only if some (G', F', X', k') is an yes-instance. By Lemma 5.3.5 and Lemma 5.3.8, we can solve each (G', F', X', k') in polynomial time. Thus, if any, a knot-free vertex deletion set of G with size at most k can be found in $2^{O(\kappa \times dfv^5)} \times n^{O(1)}$ time. $\square$

5.3.2 Taking the length of a longest directed path as aggregate parameter

Now, we investigate the length of the longest directed path $p(G)$ and $dfv(G)$ as aggregate parameters.

Lemma 5.3.10. *Given a super nice instance of Disjoint-KFVD with $F \subseteq X$, in $2^{O(dfv^3)} \times p^{O(dfv)} \times n^{O(1)}$ time, one can find (if any) a solution of size at most k.*

Proof. Let (G, F, X, k) be a super nice instance. Recall that the directed feedback vertex set F is a set of forbidden vertices ($F \subseteq X$), and G is strongly connected. If $|F| = 1$, then, for any vertex v of $V(G) \setminus F$ that can be turned into a sink, $N^+(v)$ will be a solution set for G. Therefore, the optimum solution can be found in polynomial time. Assume now that $|F| \geq 2$ and denote F by $\{v_1, \ldots, v_{dfv}\}$. As G is strongly connected, there exists a path P_1 of length at most p from v_1 to v_2 and a path P_2 of length at most p from v_2 to v_1. Denote by C the digraph $G[V(P_1) \cup V(P_2)]$; it is strongly connected, contains v_1 and v_2 and at most $2p$ vertices. Since the number of vertices in C is bounded, we may branch $2p - 1$ times (v_1 and v_2 are forbidden) by trying to guess a vertex that will be deleted in C. Each time a vertex of C will be guessed as deleted, the parameter k will decrease by one. So, k will decrease in all branches, except in the one where we guess that no vertex is deleted, and then where all the vertices of C are forbidden, which implies that no vertex of C will become sink, and also that C cannot become a knot, which would imply a removal inside C. In this case, C is a strongly connected component whose vertices are all forbidden and containing at least two vertices of F. So, we contract C to obtain a new instance G'. Formally, we remove $V(C)$ from G, add a new vertex v_C, and for all vertices of $G \setminus C$ having at least one in-neighbor (resp. out-neighbor) in C, we add an arc from v_C (resp. to v_C) to this vertex. Let F' be the set $\{v_C\} \cup F \setminus V(C)$ and notice that F' is a directed feedback vertex set of G' and that $|F'| < |F|$. Furthermore, the operation of branching plus contraction do not increase p, i.e. $p(G') \leq p(G)$. Similarly, let X' be the set $(X \setminus V(C)) \cup \{v_C\}$. Note that v_C becoming a sink in G' is equivalent to C becoming knot in G, which should be avoided in G'. If v_C has either an out-neighbor in X' or more than k out-neighbors then v_C will never become a sink after removing a feasible solution, so no additional branching is needed. Otherwise, if v_C has at most k

out-neighbors and none of them are in X' then we may branch at most another k times to guess one neighbor u of v_C that will not be removed (i.e., $u \in X'$). At this point, note that (G, F, k, X) has a solution of size at most k that does not intersect C if and only if one of these at most k instances (G', F', k, X') is a *yes*-instance. Indeed, it suffices to notice that as $V(C)$ contains only forbidden vertices in G, no vertex of C becomes a sink and C cannot become a Knot (which would imply a removal inside it); and since v_C is a forbidden vertex that cannot become sink in G', then any solution S to the KFVD problem for G is a solution of G' with some $u \in N^+(v_c)$ not in S, and conversely. Then, for each branch, we apply Branching 5.3.4 to obtain a super nice instances equivalent to (G', F', k, X'), and repeat that until either $|F'| = 1$, $k = 0$ or a solution be found.

So at each branching, either the parameter k decreases by at least one or the size of F decreases by at least one. As both values are bounded above by dfv, we branch consecutively at most $2dfv$ times. And since Branching 5.3.4 create at most 2^{2dfv^2} branches, and branching on cycle C creates at most $(2p-1) \times k$ branches, the total number of branches is $(2^{2dfv^2} \times (2p-1) \times k)^{2dfv} = 2^{O(dfv^3)} \times p^{O(dfv)}$. □

Since we obtain super nice instances in $2^{O(dfv^3)} \times n^{O(1)}$ time, the following holds.

Corollary 5.3.11. KFVD *can be solved in* $2^{O(dfv^3)} \times p^{O(dfv)} \times n^{O(1)}$ *time.*

5.3.3 On the distance to an acyclic digraph having bounded path cover

Given a directed graph $G = (V, E)$, a path cover of G is a set of directed paths such that every vertex $v \in V$ belongs to at least one path. Note that a path cover may include paths of length 0 (a single vertex). In this section, we present a simple FPT-time algorithm when we are given a special directed feedback vertex set whose removal returns an acyclic graph having bounded path cover. Recall that a path cover of an acyclic graph can be computed in polynomial time using maximum flow/matching.

Lemma 5.3.12 (Single sink along a path)**.** *Let G be a directed graph, and let $R \subseteq V(G)$ such that $G[R]$ is a DAG. Let P be any path in R. Then in a minimum knot-free vertex deletion set S of G with set of sinks Z in $G_{\bar{S}}$, we have $|Z \cap V(P)| \leq 1$.*

Proof. Assume by contradiction that $|Z \cap V(P)| \geq 2$. Let $P = (v_1, \ldots, v_p)$, and let i_1, i_2 be the indices of two consecutive vertices of $Z \cap V(P)$, or more formally such that $i_1 \leq i_2$,

$\{v_{i_1}, v_{i_2}\} \subseteq Z \cap V(P)$, and for any $i \in]i_1, i_2[$, $v_i \in V(P) \setminus Z$. Let $P' = (v_{i_1}, \ldots, v_{i_2})$ be the $v_{i_1}v_{i_2}$ subpath of P.

Let $u = N^+(v_{i_1}) \cap V(P)$. Observe that $u \neq v_{i_2}$ (as otherwise v_{i_2} would be in S and not in Z) and that $u \in S$. Let $S_{P'} = S \cap V(P')$. Observe that $S_{P'} \neq \emptyset$ as it contains u. Let v be the last (in the order of P') vertex of $S_{P'}$. Notice that $v \notin N^+(v_{i_2})$ because P is in the DAG R. Thus, we get that v_{i_2} is still a sink in $G_{\bar{S}'}$ (where $S' = S \setminus \{v\}$), and by Lemma 5.1.3 we conclude that S' is still a solution, which is a contradiction. □

Corollary 5.3.13. *Given a directed graph G, and a directed feedback vertex set F of G such that $G[V \setminus F]$ is a DAG having path cover at most c, it holds that a minimum knot-free vertex deletion set S of G can be found in time $2^{|F|} \times n^c$.*

Proof. It is well known that a minimum vertex-disjoint path cover of a DAG can be found in polynomial time. In addition, it is easy to see that a minimum path cover (where the paths may share vertices) can be found through a minimum vertex-disjoint path cover of the transitive closure of the DAG. Thus, a minimum (non-disjoint) path cover of $G[V \setminus F]$ can be found in polynomial time, and by Lemma 5.3.12 we can enumerate all possible set of sinks in time $2^{|F|} \times n^c$, which it is enough to compute a minimum knot-free vertex deletion set of G (see Corollary 5.1.4). □

5.4 On the clique-width as parameter

Recall that from Theorem 5.2.1 we can observe that KFVD is para-NP-hard concerning many digraph width measures, and it seems to be extremely hard to identify width parameters for directed graphs, which KFVD can be solved in FPT time. Next, we show that this is the case of the clique-width of the input directed graph G.

The *clique-width* of a (directed) graph G, denoted by $cw(G)$, is defined as the minimum number of labels needed to construct G, using the following four operations:

1. Create a single vertex v with an integer label ℓ (denoted by $\ell(v)$);
2. Take the disjoint union (i.e., co-join) of two graphs (denoted by $\oplus$);
3. Join by an (arc) edge every vertex labeled i to every vertex labeled j for $i \neq j$ (denoted by $\eta(i, j)$);
4. Relabel all vertices with label i by label j (denoted by $\rho(i, j)$).

An algebraic term that represents such a construction of G and uses at most k labels is said to be a *k-expression* of G (i.e., the clique-width of G is the minimum k for which G has a k-expression).

In the 90's, Courcelle proved that for every graph property Π that can be formulated in *monadic second order logic* (MSOL_1), there is an $f(k) \times n^{O(1)}$ algorithm that decides if a graph G of clique-width at most k satisfies Π (see [33, 34, 35, 38]), provided that a k-expression is given. LinEMSOL is an extension of MSOL_1 which allows searching for sets of vertices which are optimal with respect to some linear evaluation functions. Courcelle et al. [37] showed that every graph problem definable in LinEMSOL is linear-time solvable on graphs with clique-width at most k (thus, FPT when parameterized by clique-width) if a k-expression is given as input. Using a result of Oum [78], the same result follows even if no k-expression is given.

Definition 5.4.1. *[36] A directed X-path from x to y is a directed path from x to y in the subgraph induced by X.*

Proposition 5.4.2. *[36] There is a monadic second-order formula expressing the following property of vertices x, y of a set of vertices X of a directed graph G:*

"$x, y \in X$ and there is a directed X-path from x to y".

From Proposition 5.4.2 one can show that KFVD is LinEMSOL-definable. Thus Theorem 5.4.3 holds.

Theorem 5.4.3. KFVD *is FPT when parameterized by clique-width of the directed graph.*

Proof. From Proposition 5.4.2, we can construct (using shortcuts) a formula $\psi(G, S)$ such that "S is knot-free vertex deletion set of G" $\Leftrightarrow \psi(G, S)$, as follows:

$$\begin{aligned} &\exists\, Z \subset V\ [\\ &[\ \forall\, v \in Z(\ \forall\, w \in V(arc(v, w) \implies w \in S)]\ \wedge\\ &[\ \forall\, u \in \{V \setminus S\}(\ \exists\, z \in Z(\text{ there is a directed } \{V \setminus S\}\text{-path from u to z })\ ]\\ &] \end{aligned}$$

Since $\psi(G, S)$ is an MSOL_1-formula, the problem of finding $\min(S) : \psi(G, S)$ is definable in LinEMSOL. Thus we can find $\min(S)$ satisfying $\psi(G, S)$ in time $f(cw) \times n^{O(1)}$. □

The fixed-parameter tractability for clique-width parameterization implies fixed-parameter tractability of KFVD for many other popular parameters. For example, it is well-known that the clique-width of a directed graph G is at most $2^{2tw(G)+2} + 1$, where $tw(G)$ is the treewidth of the underlying undirected graph (see [36, Proposition 2.114]). However, although Theorem 5.4.3 implies the FPT-membership of the problem parameterized by the treewidth of the underlying undirected graph, the dependence on $tw(G)$ provided by the model checking framework is huge. So, it is still pertinent to ask whether such a parameterized problem admits a more efficient algorithm, which is discussed in Section 5.5.

5.5 On the treewidth as parameter

Given a tree decomposition $\mathcal{T}$, we denote by t one node of $\mathcal{T}$ and by X_t the set of vertices contained in the *bag* of t. We assume, without loss of generality, that $\mathcal{T}$ is a *nice* tree decomposition (see [41]), that is, we assume that there is a special root node r such that $X_t = \emptyset$ and all edges of the tree are directed towards r and each node t has one of the following four types: *Leaf*, *Introduce vertex*, *Forget vertex*, and *Join*.

Based on the following results, we can assume that we are given a nice tree decomposition of G.

Theorem 5.5.1. *[14] There exists an algorithm that, given an n-vertex graph G and an integer k, runs in time $2^{O(k)} \times n$ and either outputs that the treewidth of G is larger than k or constructs a tree decomposition of G of width at most $5k + 4$.*

Lemma 5.5.2. *[41] Given a tree decomposition $(T, \{X_t\}_{t\in V(T)})$ of G of width at most k, one can in time $O(k^2 \cdot \max(|V(T)|, |V(G)|))$ compute a nice tree decomposition of G of width at most k that has at most $O(k|V(G)|)$ nodes.*

Now we are ready to use a nice tree decomposition in order to obtain an FPT-time algorithm with single exponential dependency on $tw(G)$ and linear with respect to n.

Theorem 5.5.3. Knot-Free Vertex Deletion *can be solved in $2^{O(tw \log tw)} \times n$ time, but assuming ETH there is no $2^{o(tw)}n^{O(1)}$ time algorithm for* KFVD, *where tw is the treewith of the underlying undirected graph of the input G.*

Proof. Let $\mathcal{T} = (T, \{X_t\}_{t\in V(T)})$ be a nice tree decomposition of the input digraph G, with width equal to tw. First, we consider the following additional notation and definitions: t

is the index of a bag of T; G_t is the graph induced by all vertices $v \in X_{t'}$ such that either $t' = t$ or $X_{t'}$ is a descendant of X_t in T;

Given a knot-free vertex deletion set S of G, for any bag X_t there is a partition of X_t into

$$S_t, Z_t, F_t, B_t$$

such that

- S_t (removed) is the set of vertices of X_t that are going to be removed ($S_t = S \cap X_t$);
- Z_t (sinks) is the set of vertices of X_t that are going to be turned into sinks after the removal of S;
- F_t (free/released) is the set of vertices of X_t that, after the removal of S, are going to reach a sink in the graph G_t;
- B_t (blocked) is the set of vertices of X_t that, after the removal of S, are going to reach *no* sink in the graph G_t.

Let $G \setminus S$ be the graph resulting from the removal of S. Each vertex of F_t reaches a sink in $G_t \setminus S$ through a path that either goes through no other vertex of F_t or traverses some other vertices of F_t before it reaches a sink. Therefore, from a solution S we can define a graph H_{F_t}, where $V(H_{F_t}) = F_t$ and each vertex v of H_{F_t} has at most one out-edge, which representing the existence in G_t of path between v to another vertex u that will be used to v reaches a sink. Actually, v can reach several vertices of F_t, but for the problem in question, it is enough to be able to recover a path P from v to a sink, and we can assume that u is the vertex of F_t closest to v in P.

Similarly, each vertex of B_t reaches a sink in $G \setminus S$ through a path that either goes through no other vertex of B_t or traverses some other vertices of B_t. In the latter case, G_t may already contain the subpath between a blocked vertex v to another blocked vertex u that will be used to release v in the future. Therefore, from a solution S we also can define a graph H_{B_t} where $V(H_{B_t}) = B_t$ and each vertex v of H_{B_t} has at most one out-edge, representing the existence in G_t of path between v to another blocked vertex u that will be used to v reaches a sink.

Note that both H_{F_t} and H_{B_t} are DAGs with maximum out-degree at most one, so each sink roots an arborescence converging to the root, and you can enumerate all possible pairs H_{F_t}, H_{B_t} in time $2^{O(tw \log tw)}$.

The recurrence relation of our dynamic programming has the signature

$$C[t, S_t, Z_t, F_t, B_t, H_{F_t}, H_{B_t}],$$

representing the minimum number of vertices in G_t that must be removed in order to produce a graph such that for every remaining vertex v either v reaches a vertex that became a sink (possibly the vertex itself), or v reaches a vertex in B_t (meaning that it may still be released in the future), where

- the vertices in S_t must be removed;
- the vertices in Z_t will become sink;
- every vertex in F_t will have a path to a sink in the resulting graph;
- no vertex in B_t will have a path to a sink in G_t in the resulting graph;
- for each sink v of H_{F_t} there must be a path from v to a sink of the resulting graph, which traverses no other vertex of F_t;
- for every edge vu in H_{F_t} or H_{B_t} there must be a path from v to u in the resulting graph;
- S_t, Z_t, F_t, B_t form a partition of X_t;
- H_{F_t} and H_{B_t} are DAGs with maximum out-degree at most one, with $V(H_{F_t}) = F_t$ and $V(H_{B_t}) = B_t$.

Notice that the generated table has size $4^{tw} \times 2.tw^{tw} \times tw \times n$, and when $t = r$, $X_t = \emptyset$ and therefore $C[r, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset]$ must contain the size of a minimum knot-free vertex deletion set of $G_r = G$.

The recurrence relation for each type of node is described as follows.

First, notice that if $v \in Z_t$ and there is an out-neighbor $w \in X_t$ of v that is not in S_t, there is an inconsistency, i.e. w must be deleted. In addition, if $v \in B_t$ but has an out-neighbor in $Z_t \cup F_t$, there is another inconsistency (v is not blocked), if $v \in F_t$ but the removal of $S_t \cup B_t$ turns v into an isolated vertex, v is not released and it must belong to B_t, and if $v \in F_t$ has degree one in H_{F_t} but is an in-neighbor of a vertex in Z_t there is also an inconsistency (v must be a sink in H_{F_t}).

For the inconsistent cases, $C[t, S_t, Z_t, F_t, B_t, H_{F_t}, H_{B_t}] = +\infty$. Such cases can be recognized and treated by simple preprocessing in linear time on the size of the table. Therefore, we consider next only consistent cases.

Leaf Node: If X_t is a leaf node then $X_t = \emptyset$. Therefore

$$C[t, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset] = 0.$$

Introduce Node: Let X_t be a node of T with a child $X_{t'}$ such that $X_t = X_{t'} \cup \{v\}$ for some $v \notin X_{t'}$. We have $C[t, S_t, Z_t, F_t, B_t, H_{F_t}, H_{B_t}]$ equal to:

$$\begin{cases}
1)\ \textit{case } v \in S_t: \\
-\quad C[t', S_t \setminus \{v\}, Z_t, F_t, B_t, H_{F_t}, H_{B_t}] + 1, \\
2)\ \textit{case } v \in Z_t: \\
-\quad C[t', S_t, Z_t \setminus \{v\}, F_t \setminus A, B_t \cup A, H_{F_t}[F_t \setminus A], H_{B_t} \cup H_{F_t}[A]], \\
\text{where } A = \{w \in F_t : w \text{ reaches a sink } u \text{ (possibly } w = u\text{) in } H_{F_t} \text{ s.t. } uv \in E(G)\}, \\
3)\ \textit{case } v \in F_t: \\
-\quad C[t, S_t, Z_t, F_t \setminus \{A' \cup \{v\}\}, B_t \cup A', H_{F_t}[F_t \setminus \{A' \cup \{v\}\}], H_{B_t} \cup H_{F_t}[A']], \\
\text{where } A' = \{w \in F_t : w \text{ reaches } v \text{ in } H_{F_t}\}, \\
4)\ \textit{case } v \in B_t: \\
-\quad C[t', S_t, Z_t, F_t, B_t \setminus \{v\}, H_{F_t}, H_{B_t}[B_t \setminus \{v\}]].
\end{cases}$$

Recall that, for simplicity, we consider only consistent cases, thus if a vertex of H_{F_t} has out-neighbors in Z_t then it is a sink in H_{F_t}. In addition, in case 2 it holds that $N^+(v) \cap X_t \subseteq S_t$, in case 3 it holds that $N^+(v) \cap (Z_t \cup F_t) \neq \emptyset$, and in case 4 it holds that $N^+(v) \cap \{Z_t \cup F_t\} = \emptyset$. Also note that, A and A' represent set of vertices that will be released through paths that traverses v.

Forget Node: Let X_t be a forget node with a child $X_{t'}$ such that $X_t = X_{t'} \setminus \{v\}$, for some $v \in X_{t'}$. The forget node selects the best scenario considering all the possibilities for the forgotten vertex, discarding cases that lead to non-feasible solutions. In this problem, unfeasible cases are identified when the forgotten vertex $v \in X_{t'}$ was blocked and is a sink in H_{B_t} (it reaches no other node in B_t that can release it in the future). Hence:

$$C[t, S_t, Z_t, F_t, B_t, H_{F_t}, H_{B_t}] = \min \begin{cases} C[t', S_t \cup \{v\}, Z_t, F_t, B_t, H_{F_t}, H_{B_t}], \\ C[t', S_t, Z_t \cup \{v\}, F_t, B_t, H_{F_t}, H_{B_t}], \\ \forall_{H'_{F_t}} C[t', S_t, Z_t, F_t \cup \{v\}, B_t, H'_{F_t}, H_{B_t}], \\ \forall_{H'_{B_t}} C[t', S_t, Z_t, F_t, B_t \cup \{v\}, H_{F_t}, H'_{B_t}] \end{cases}$$

where H'_{F_t} and H'_{B_t} are graphs such that

> H_{F_t} can be obtained from H'_{F_t} by removing v and adding edges from every in-neighbor of v in H'_{F_t} to the out-neighbor of v in H'_{F_t}, if any.
>
> H_{B_t} can be obtained from H'_{B_t} by removing v and adding edges from every in-neighbor of v in H'_{F_t} to the out-neighbor of v in H'_{F_t}, where v is not a sink of H'_{F_t}.

Join Node: Let X_t be a join node with children X_{t_1} and X_{t_2}, such that $X_t = X_{t_1} = X_{t_2}$. For any optimal knot-free vertex deletion set S of G it holds that $V(G_t) \cap S = \{V(G_{t_1}) \cap S\} \cup \{V(G_{t_2}) \cap S\}$. Clearly, if $S_t \subseteq S$ then we can assume that $S_t = S_{t_1} = S_{t_2}$. In addition, $Z_t = Z_{t_1} = Z_{t_2}$ otherwise we will have an inconsistency. Also note that a vertex is released in G_t if it reaches a vertex (possibly the vertex itself) that is released either in G_{t_1} or G_{t_2}.

Thus $C[t, S_t, Z_t, F_t, B_t, H_{F_t}, H_{B_t}]$ is equal to:

$$\min_{\forall \text{ tuple } (F_{t_1}, F_{t_2}, B_{t_1}, B_{t_2} H_{F_{t_1}}, H_{F_{t_2}}, H_{B_{t_1}}, H_{B_{t_2}})} \left\{ \begin{array}{c} C[t_1, S_t, Z_t, F_{t_1}, B_{t_1}, H_{F_{t_1}}, H_{B_{t_1}}] \\ + \\ C[t_2, S_t, Z_t, F_{t_2}, B_{t_2}, H_{F_{t_2}}, H_{B_{t_2}}] \end{array} \right\} - |S_t|,$$

where

1. each vertex in $F_{t_i} \cap B_{t_j}$ is a sink of $H_{B_{t_j}}$;
2. $F_t = F_{t_1} \cup F_{t_2} \cup \{w \in B_{t_j} : w$ reaches a sink u in $H_{B_{t_j}}$ s.t. $u \in F_{t_i} \cap B_{t_j}\}$;
3. $H_{F_t}[F_{t_1} \cap F_{t_2}] = H_{F_{t_1}}[F_{t_1} \cap F_{t_2}] = H_{F_{t_2}}[F_{t_1} \cap F_{t_2}]$;
4. $(E(H_{B_{t_1}}) \cup E(H_{B_{t_2}})) \setminus E(H_{B_t}) \subseteq E(H_{F_t})$;

5. each vertex has at most one edge in $E(H_{B_{t_1}}) \cup E(H_{B_{t_2}})$;

6. $E(H_{B_t}) \subseteq E(H_{B_{t_1}}) \cup E(H_{B_{t_2}})$.

Note that in (2) it is described that F_t is the set of vertices that either are released in G_{t_i} ($i \in \{1, 2\}$) or can be released in G_t by vertices of $F_{t_1} \cup F_{t_2}$, even if they are blocked in both G_{t_1} and G_{t_2}; this can occur, for example, if a blocked vertex v reaches another blocked vertex w in G_{t_1}, and in G_{t_2} the vertex w is released. The rest of the restrictions only provide a description of the tuples that actually need to be considered.

Now, in order to run the algorithm, one can visit the bags of $\mathcal{T}$ in a bottom-up fashion, performing the queries described for each type of node. Thus, one can fill each entry of the table in $2^{O(tw \log tw)}$ time, and as the table has size $2^{O(tw \log tw)} \times n$, the dynamic programming can be performed in time $2^{O(tw \log tw)} \times n$.

Regarding correctness, let S^* be a minimum knot-free vertex deletion set of a digraph G with a tree decomposition $\mathcal{T}$. Let $S_t^*, Z_t^*, F_t^*, B_t^*$ be a partition of the vertices of X_t into removed, sinks, released and blocked, with respect to G_t after the removal of S^*. Note that $S_t^* = X_t \cap S^*$, and S^* and naturally defines graphs $H_{F_t}^*, H_{B_t}^*$. The following lemma holds.

Lemma 5.5.4. *Let $\widehat{S}$ be a set for which the minimum is attained in the definition of $C[t, S_t^*, Z_t^*, F_t^*, B_t^*, H_{F_t}^*, H_{B_t}^*]$. Then $S = \widehat{S} \cup (S^* \setminus V(G_t))$ is also a solution (which is minimum) for* KFVD.

Proof. Suppose that $S = \widehat{S} \cup (S^* \setminus V(G_t))$ is not a solution for KFVD. Then there is a vertex v of B_t^* that does not reach a sink in $G \setminus S$; otherwise, there is a contradiction. Since $G_t \setminus S$ preserves the paths represented by edges in $H_{B_t}^*$, without loss of generality, we can assume that v is a sink. Since v is sink in $H_{B_t}^*$, in the graph $G \setminus S^*$ the vertex v reaches a sink through a path P that either tranverses no other vertex of G_t (and such a path P is preserved in $G \setminus S$), or it reaches some vertices of F_t^*. In the second case, the subpath of P from v to the closest vertex of $F_t \cap P$, say w, is also preserved, since it does not traverse any vertex of G_t. Then v reaches w in $G \setminus S$, and as the paths represented by edges in $H_{F_t}^*$ is also preserved in $G_t \setminus S$, then v reaches a sink of $H_{F_t}^*$ in $G \setminus S$. Finally, by definition each sink of $H_{F_t}^*$ has a path to a sink in $G_t \setminus \widehat{S}$, thus v reaches a sink in $G \setminus S$. Therefore, this vertex v does not exist and $S = \widehat{S} \cup (S^* \setminus V(G_t))$ is a solution for KFVD. Since $|\widehat{S}| \leq |S^* \setminus V(G_t)|$ then S is an optimal solution. $\square$

Fact 1 implies that we have stored enough information. At this point, the correctness of the recursive formulas is straightforward from their descriptions.

Finally, to show a lower bound based on ETH, using the polynomial-time reduction that preserves parameter presented in Theorem 4.2.2, we obtain in polynomial time a graph with $|V| = 2n + 2m$, and so $tw = O(n + m)$. Therefore, if KFVD can be solved in $2^{o(tw)} \times |V|^{O(1)}$ time, then we can solve 3-CNF-SAT in $2^{o(n+m)} \times (n + m)^{O(1)}$ time, i.e., ETH fails. This conclude the proof of Theorem 5.5.3. □

Next, we improve the result presented in the Theorem 4.2.3 by including the treewidth as an additional parameter.

Corollary 5.5.5. *Unless $NP \subseteq coNP/poly$, $[k, tw]$-KFVD does not admit a polynomial kernel, even when a largest SCC of the input graph G has size 2.*

Proof. It is enough to show that the underlying undirected graph of the instance G' constructed in Theorem 4.2.3 has treewidth at most $|R|$. Consider the following bags:

- a root bag $X_r = \{c_i^1 |\ v_i \in R\}$;
- a bag $X_i = \{c_i^1, c_i^2, c_i^3, c_i^4\}\ \forall\ v_i \in R$;
- a bag $X_e = X_r \cup \{w\}$, for every arc $e = (w, c_i^1)$ of G';
- and a bag $\{u |\ u \in C_j^\ell\}\ \forall\ 1 \leq \ell \leq k' + 1$.

Clearly it is possible to construct a tree decomposition T with the bags described above (see Figure 5.5), thus $tw(G') \leq |R|$. □

5.6 Conclusions

In this chapter, we consider directed width parameterizations for KFVD. We proved that k-KFVD remains W[1]-hard even when the input graph has K-width equals 2 and the longest directed path of size 5. From the above result, we can observe that KFVD is para-NP-hard with respect to several well-known width measures. In addition, we show that KFVD parameterized by cliquewidth of the directed graph is FPT, and we proposed two FPT-algorithms, each exploring additional parameters to the *directed feedback vertex set number* (dfv). The first one, combining dfv with K-width (κ), runs in $2^{O(\kappa dfv^5)} n^{O(1)}$

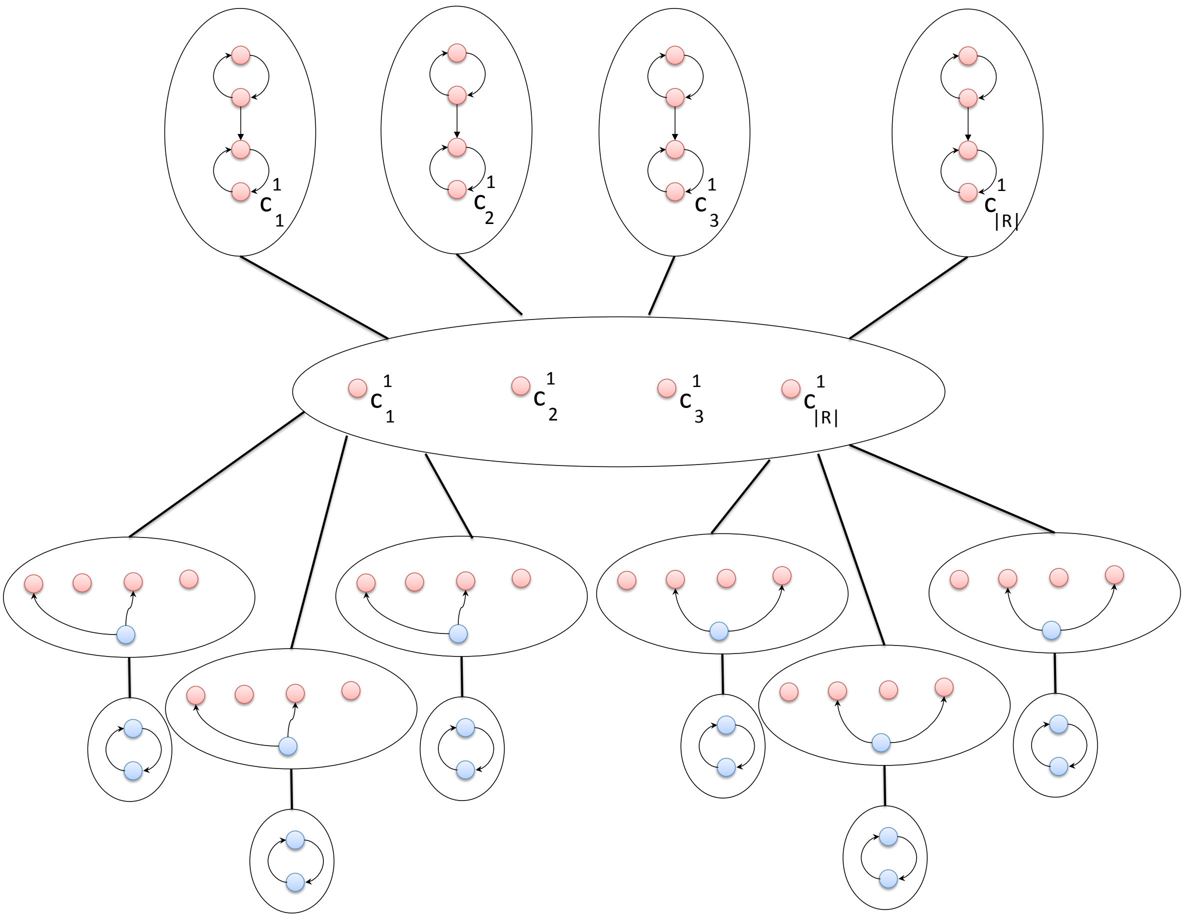

Figure 5.5: A tree decomposition of G'.

time. The second one, combining dfv with the length of the longest directed path p, runs in $2^{O(dfv^3)}\ p^{O(dfv)}n^{O(1)}$ time. An FPT-time algorithm is presented when we are given a special directed feedback vertex set whose removal returns an acyclic graph having path cover bounded by a constant c. Also, we proved that: KFVD can be solved in FPT time when parameterized by cliquewidth of the underlying undirected graph. Finally, KFVD can be solved in time $2^{O(tw \log tw)} \times n$, but assuming ETH it cannot be solved in $2^{o(tw)} \times n^{O(1)}$, where tw is the treewidth of the underlying undirected graph and unless $NP \subseteq coNP/poly$, KFVD parameterized by the size of the solution k and the treewidth of the underlying graph does not admit a polynomial kernel, even when the largest SCC of the input graph G has size 2.

Chapter 6

Final Remarks and Future Works

In this chapter, we make remarks about KFVD problem and its similarities to the DFVS problem in order to point directions to the parameterized analysis of KFVD. We start by comparing the deadlock characterization in the OR and AND models:

Deadlock in the OR-model – the occurrence of deadlocks in wait-for graphs G working according to the OR-model is characterized by the existence of knots in G [11, 59]. A knot in a directed graph G is a strongly connected subgraph Q of G, such that $|V(Q)| \geq 2$ and no vertex in $V(Q)$ is an in-neighbour of a vertex in $V(G) \setminus V(Q)$. Given a graph G and a positive integer k, the KFVD problem consists of determining whether there exists a subset $S \subseteq V(G)$ of size at most k such that $G[V \setminus S]$ is knot-free. This problem was proved to be NP-hard in [24].

Deadlock in the AND-model – the occurrence of deadlocks in wait-for graphs G working according to the AND-model is characterized by the existence of cycles in G [11, 8]. Thus, given a graph G and a positive integer k, the problem of determining whether there exists a subset $S \subseteq V(G)$ of size at most k such that $G[V \setminus S]$ is cycle-free is the well-known Directed Feedback Vertex Set (DFVS) problem, proved to be NP-hard in the seminal paper of Karp [67], and proved to be fixed-parameter tractable in [28].

We define λ–Deletion($\mathbb{M}$) as a generic optimization problem for deadlock resolution, where λ indicates the type of deletion operation to be used in order to break all the deadlocks, and $\mathbb{M} \in$ {AND, OR, X-Out-Of-Y, AND-OR} is the deadlock model of the input wait-for graph G. Vertex–Deletion(AND) and Arc–Deletion(AND) are equivalent to Directed Feedback Vertex Set and Directed Feedback Arc Set, respectively. We proved that Arc–Deletion(OR) and Output–Deletion(OR) are solvable in polynomial time. In addition, KFVD was shown to be NP-complete. Such results are summarized in the Table 6.1.

λ–Deletion($\mathbb{M}$)

$\lambda \setminus \mathbb{M}$	AND	OR	AND-OR	X-Out-Of-Y
Arc	NP-H	P	NP-H	NP-H
Vertex	NP-H	NP-H	NP-H	NP-H
Output	NP-H	P	NP-H	NP-H

Table 6.1: Computational complexity of λ–Deletion($\mathbb{M}$).

A study of the complexity of KFVD in different graph classes was also done. We proved that the problem remains NP-hard even for strongly connected graphs and planar bipartite graphs with maximum degree four. Furthermore, we proved that for graphs with maximum degree three the problem can be solved in polynomial time. Thus we have the Table 6.2:

KFVD

Instance	Complexity
Weakly connected	NP-Hard
Strongly connected	NP-Hard
Planar, bipartite, $\Delta(G) \geq 4$ and $\Delta(G)^+ = 2$	NP-Hard
$\Delta(G) = 3$	Polynomial
$\Delta(G) = 2$	Trivial
$\Delta(G)^+ = 1$	Trivial

Table 6.2: Complexity of KFVD for some graph classes.

In addition, we explored weighted wait-for graphs, where we show that W–λ–Deletion(OR) can be reduced into λ–Deletion(OR) and W–Arc–Deletion(OR) can also be solved in linnear time. W–λ–Deletion(AND) can be reduced into λ–Deletion(AND).

In chapter 4, we study the Knot-Free Vertex Deletion problem from a parameterized complexity point of view. First, we proved that KFVD with the natural parameter k is W[1]-hard. Next, we consider φ, the maximum size of an SCC of the input directed graph, as an additional parameter. We show that KFVD can be solved in $2^{k \log \varphi} n^{O(1)}$ time and unless SETH fails it cannot be solved in $(2-\epsilon)^{(k \log \varphi)} n^{O(1)}$ time. Also, we remark that k-KFVD has no polynomial kernel even if the input graph has only SCC's with size bounded by 2. After that, we present an algorithm that runs in $2^{\phi} n^{O(1)}$ time, which it is appropriate for directed graphs where there are few vertices, ϕ, with out-degree at most k. In addition, assuming ETH, we show that KFVD cannot be solved in $2^{o(\phi)} n^{O(1)}$ time. Considering the treewidth of the underlying graph tw as parameter, we show that KFVD can be solved in $2^{O(tw \log tw)} n^{O(1)}$ time, but assuming ETH it cannot be solved in $2^{o(tw)} n^{O(1)}$ time.

Table 6.3 summarizes the fine-grained parameterized complexity analysis presented in this work.

Table 6.3: Fine-grained parameterized complexity of Knot-Free Vertex Deletion.

		Complexity	Running time	Lower bounds assuming (S)ETH
Parameter	k	W[1]-hard	n^k	no $f(k) \times n^{o(k)}$ alg.
	k, φ	FPT	$2^{k \log \varphi} \times n^{O(1)}$	no $(2-\epsilon)^{k \log \varphi} \times n^{O(1)}$ alg.
	ϕ	FPT	$2^{\phi} \times n^{O(1)}$	no $2^{o(\phi)} \times n^{O(1)}$ alg.
	tw	FPT	$2^{O(tw \log tw)} \times n^{O(1)}$	no $(2)^{o(tw)} \times n^{O(1)}$ alg.

In chapter 5, a parameterized complexity study of KFVD on directed width measures is also done. We proved that KFVD with the natural parameter k even when the input graph has K-width 2 and the longest directed path is 5 is also W[1]-hard. From the above result, we can observe that KFVD is para-NP-hard with respect to several well-known width measures. In addition, we show that KFVD parameterized by cliquewidth of the directed graph is FPT, and we proposed two FPT-algorithms, each exploring additional parameters to the *directed feedback vertex set number* (dfv). The first one, combining dfv with K-width (κ), runs in $2^{O(\kappa dfv^5)} n^{O(1)}$ time. The second one, combining dfv with the length of the longest directed path p, runs in $2^{O(dfv^3)} p^{O(dfv)} n^{O(1)}$ time. Finally, an FPT-time algorithm is presented when we are given a special directed feedback vertex set whose removal returns an acyclic graph having path cover bounded by a constant c.

Recall that knots characterize the presence of deadlock. So, the algorithms presented in this work have also practical value. The most common approach to deal with deadlock is to forbid the formation of cycles in the directed graph as the computation proceeds. This approach, although simple and easy to implement, is very restrictive. Having an algorithm that breaks the knots of a graph (therefore removing deadlocks) in exponential time, but over a controlled characteristic, allows the construction of a more permissive deadlock prevention. For example, as Algorithm 2 is FPT with respect to k and the size of the largest SCC in G, it is possible to forbid only the formation of large knots, rather than cycles.

The KFVD problem is closely related to the DFVS problem not only because of their relation with deadlocks, but some structural similarities between them: the goal of DFVS is to obtain a directed acyclic graph (DAG) via vertex deletion (in such graphs *all* maximal directed paths end in a sink); the goal of KFVD is to obtain a knot-free graph, and in such graphs for every vertex v there *exists* at least one maximal path containing v that ends in a sink. Finally, every directed feedback vertex set is a knot-free vertex deletion set; thus, the size of a minimum directed feedback vertex set is an upper bound for KFVD. Besides, the DFVS problem is closely related to the KFVD problem and indicates some closeness with the sought solution of KFVD. Hence, the DFVS-number is unquestionably a natural parameter to be explored considering that it can be obtained in FPT-time. Finally, we leave two open questions:

- Can dfv-KFVD be solved in FPT time?
- Given a minimum directed feedback vertex set F, can KFVD be solved in $f(dfv, c) \times n^{O(1)}$ time, when c is the path cover of $G[V \setminus F]$?

References

[1] AKHOONDIAN AMIRI, S.; KAISER, L.; KREUTZER, S.; RABINOVICH, R.; SIEBERTZ, S. Graph searching games and width measures for directed graphs. In *32nd International Symposium on Theoretical Aspects of Computer Science (STACS 2015)* (2015), Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

[2] ALBER, J.; FELLOWS, M. R.; NIEDERMEIER, R. Polynomial-time data reduction for dominating set. *Journal of the ACM (JACM) 51*, 3 (2004), 363–384.

[3] ATREYA, R.; MITTAL, N.; KSHEMKALYANI, A. D.; GARG, V. K.; SINGHAL, M. Efficient detection of a locally stable predicate in a distributed system. *Journal of Parallel and Distributed Computing 67*, 4 (2007), 369–385.

[4] BANSAL, S.; HOTA, C. Priority-based job scheduling in distributed systems. In *International Conference on Information Systems, Technology and Management* (2009), Springer, pp. 110–118.

[5] BARÁT, J. Directed path-width and monotonicity in digraph searching. *Graphs and Combinatorics 22*, 2 (2006), 161–172.

[6] BARBOSA, V. C. *Massively Parallel Models of Computation: Distributed Parallel Processing in Artificial Intelligence and Optimisation*. Ellis Horwood, 1993.

[7] BARBOSA, V. C. *An Introduction to Distributed Algorithms*. MIT Press, 1996.

[8] BARBOSA, V. C. The combinatorics of resource sharing. In *Models for Parallel and Distributed Computation*. Springer, 2002, pp. 27–52.

[9] BARBOSA, V. C.; BENEVIDES, M. R. A graph-theoretic characterization of AND-OR deadlocks. Tech. Rep. COPPE-ES-472/98, Federal University of Rio de Janeiro, Rio de Janeiro, Brazil, 1998.

[10] BARBOSA, V. C.; BENEVIDES, M. R.; OLIVEIRA FILHO, A. L. A priority dynamics for generalized drinking philosophers. *Information Processing Letters 79*, 4 (2001), 189–195.

[11] BARBOSA, V. C.; CARNEIRO, A. D. A.; PROTTI, F.; SOUZA, U. S. Deadlock models in distributed computation: Foundations, design, and computational complexity. In *Proceedings of the 31st ACM/SIGAPP Symposium on Applied Computing* (2016), pp. 538–541.

[12] BERWANGER, D.; DAWAR, A.; HUNTER, P.; KREUTZER, S.; OBDRŽÁLEK, J. The dag-width of directed graphs. *Journal of Combinatorial Theory, Series B 102*, 4 (2012), 900 – 923.

[13] Berwanger, D.; Grädel, E. Entanglement – a measure for the complexity of directed graphs with applications to logic and games. In *Logic for Programming, Artificial Intelligence, and Reasoning* (Berlin, Heidelberg, 2005), F. Baader and A. Voronkov, Eds., Springer Berlin Heidelberg, pp. 209–223.

[14] Bodlaender, H. L.; Drange, P. G.; Dregi, M. S.; Fomin, F. V.; Lokshtanov, D.; Pilipczuk, M. A $c^k n$ 5-approximation algorithm for treewidth. *SIAM Journal on Computing 45*, 2 (2016), 317–378.

[15] Bodlaender, H. L.; Penninkx, E. A linear kernel for planar feedback vertex set. In *International Workshop on Parameterized and Exact Computation* (2008), Springer, pp. 160–171.

[16] Bodlaender, H. L.; Penninkx, E.; Tan, R. B. A linear kernel for the k-disjoint cycle problem on planar graphs. In *International Symposium on Algorithms and Computation* (2008), Springer, pp. 306–317.

[17] Bokal, D.; Brešar, B.; Jerebic, J. A generalization of Hungarian method and Hall's theorem with applications in wireless sensor networks. *Discrete Applied Mathematics 160*, 4 (2012), 460–470.

[18] Bondy, J. A.; Murty, U. S. R. *Graph theory with applications*, vol. 290. Macmilan, 1976.

[19] Bonsma, P. S.; Brueggemann, T.; Woeginger, G. J. A faster fpt algorithm for finding spanning trees with many leaves. In *International Symposium on Mathematical Foundations of Computer Science* (2003), Springer, pp. 259–268.

[20] Bracha, G.; Toueg, S. Distributed deadlock detection. *Distributed Computing 2*, 3 (1987), 127–138.

[21] Brzezinski, J.; Helary, J.-M.; Raynal, M.; Singhal, M. Deadlock models and a general algorithm for distributed deadlock detection. *Journal of parallel and distributed computing 31*, 2 (1995), 112–125.

[22] Cai, L.; Juedes, D. On the existence of subexponential parameterized algorithms. *Journal of Computer and System Sciences 67*, 4 (2003), 789–807.

[23] Calabro, C.; Impagliazzo, R.; Paturi, R. The complexity of satisfiability of small depth circuits. In *International Workshop on Parameterized and Exact Computation* (2009), Springer, pp. 75–85.

[24] Carneiro, A. D. A.; Protti, F.; Souza, U. S. Deletion graph problems based on deadlock resolution. In *Computing and Combinatorics - 23rd International Conference, COCOON 2017, Hong Kong, China, August 3-5, 2017, Proceedings* (2017), pp. 75–86.

[25] Chahar, P.; Dalal, S. Deadlock resolution techniques: An overview. *International Journal of Scientific and Research Publications 3*, 7 (2013).

[26] Chandy, K. M.; Lamport, L. Distributed snapshots: determining global states of distributed systems. *ACM Transactions on Computer Systems 3* (1985), 63–75.

[27] Chandy, K. M.; Misra, J.; Haas, L. M. Distributed deadlock detection. *ACM Transactions on Computer Systems (TOCS) 1*, 2 (1983), 144–156.

[28] Chen, J.; Liu, Y.; Lu, S.; O'sullivan, B.; Razgon, I. A fixed-parameter algorithm for the directed feedback vertex set problem. *Journal of the ACM (JACM) 55*, 5 (2008), 21.

[29] Chen, J.; Meng, J. On parameterized intractability: Hardness and completeness. *The Computer Journal 51*, 1 (2007), 39–59.

[30] Coffman, E. G.; Elphick, M.; Shoshani, A. System deadlocks. *ACM Computing Surveys (CSUR) 3*, 2 (1971), 67–78.

[31] Cook, S. A. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing* (1971), ACM, pp. 151–158.

[32] Cormen, T. H.; Leiserson, C. E.; Rivest, R. L.; Stein, C. *Introduction to Algorithms*. MIT press, 2009.

[33] Courcelle, B. The monadic second-order logic of graphs. I. recognizable sets of finite graphs. *Information and Computation 85*, 1 (1990), 12–75.

[34] Courcelle, B. The expression of graph properties and graph transformations in monadic second-order logic. *Handbook of Graph Grammars 1* (1997), 313–400.

[35] Courcelle, B.; Engelfriet, J. *Graph Structure and Monadic Second-Order Logic: A Language-Theoretic Approach*, 1st ed. Cambridge University Press, 2012.

[36] Courcelle, B.; Engelfriet, J. *Graph structure and monadic second-order logic: a language-theoretic approach*, vol. 138. Cambridge University Press, 2012.

[37] Courcelle, B.; Makowsky, J. A.; Rotics, U. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory of Computing Systems 33*, 2 (2000), 125–150.

[38] Courcelle, B.; Mosbah, M. Monadic second-order evaluations on tree-decomposable graphs. *Theoretical Computer Science 109*, 1–2 (1993), 49–82.

[39] Crampton, J.; Gutin, G.; Watrigant, R. A multivariate approach for checking resiliency in access control. In *International Conference on Algorithmic Applications in Management* (2016), Springer, pp. 173–184.

[40] Cristian, F.; Fetzer, C. The timed asynchronous distributed system model. *IEEE Transactions on Parallel and Distributed Systems 10*, 6 (1999), 642–657.

[41] Cygan, M.; Fomin, F. V.; Kowalik, Ł.; Lokshtanov, D.; Marx, D.; Pilipczuk, M.; Pilipczuk, M.; Saurabh, S. *Parameterized algorithms*, vol. 3. Springer, 2015.

[42] Datta, A.; Ghosh, S. Synthesis of a class of deadlock-free petri nets. *Journal of the ACM (JACM) 31*, 3 (1984), 486–506.

[43] DE MENDÍVIL, J. G.; FARIÑA, F.; GARITAGOTIA, J. R.; ALASTRUEY, C. F.; BERNABEU-AUBAN, J. M. A distributed deadlock resolution algorithm for the and model. *IEEE Transactions on Parallel and Distributed Systems 10*, 5 (1999), 433–447.

[44] DING, Y.; HE, Y.; JIANG, J. Multi-robot cooperation method based on the ant algorithm. In *Swarm Intelligence Symposium, 2003. SIS'03. Proceedings of the 2003 IEEE* (2003), IEEE, pp. 14–18.

[45] DOM, M.; LOKSHTANOV, D.; SAURABH, S. Kernelization lower bounds through colors and ids. *ACM Transactions on Algorithms (TALG) 11*, 2 (2014), 13.

[46] DOS SANTOS, V. F.; DOS SANTOS SOUZA, U. Uma introdução à complexidade parametrizada. *Anais da 34º Jornada de Atualização em Informática, CSBC* (2015), 232–273.

[47] DOS SOUZA, U. *Multivariate investigation of NP-hard problems: Boundaries between parameterized tractability and intractability.* Tese de Doutorado, Fluminense Federal University, Niterói-RJ, Brazil, 2014.

[48] DOWNEY, R. G.; FELLOWS, M. R. Parameterized complexity. *Monogr. Comput. Sci. Springer, New York 87* (1999).

[49] DOWNEY, R. G.; FELLOWS, M. R. *Parameterized complexity.* Springer Science & Business Media, 2012.

[50] DOWNEY, R. G.; FELLOWS, M. R. *Fundamentals of parameterized complexity*, vol. 4. Springer, 2013.

[51] FELLOWS, M. R. Parameterized complexity: the main ideas and some research frontiers. In *International Symposium on Algorithms and Computation* (2001), Springer, pp. 291–307.

[52] FLUM, J.; GROHE, M. Parameterized complexity theory, volume xiv of texts in theoretical computer science. an eatcs series, 2006.

[53] GALČÍK, F.; KATRENIČ, J.; SEMANIŠIN, G. On computing an optimal semi-matching. In *Graph-Theoretic Concepts in Computer Science* (2011), Springer, pp. 250–261.

[54] GANIAN, R.; HLINĚNÝ, P.; KNEIS, J.; LANGER, A.; OBDRŽÁLEK, J.; ROSSMANITH, P. Digraph width measures in parameterized algorithmics. *Discrete applied mathematics 168* (2014), 88–107.

[55] GANIAN, R.; HLINĚNÝ, P.; KNEIS, J.; MEISTER, D.; OBDRŽÁLEK, J.; ROSSMANITH, P.; SIKDAR, S. Are there any good digraph width measures? In *International Symposium on Parameterized and Exact Computation* (2010), Springer, pp. 135–146.

[56] GARY, M. R.; JOHNSON, D. S. Computers and intractability: A guide to the theory of np-completeness, 1979.

[57] Gruber, H. Digraph complexity measures and applications in formal language theory. *Discrete Mathematics & Theoretical Computer Science 14*, 2 (2012), 189–204.

[58] Guo, J.; Niedermeier, R. Invitation to data reduction and problem kernelization. *ACM SIGACT News 38*, 1 (2007), 31–45.

[59] Holt, R. C. Some deadlock properties of computer systems. *ACM Computing Surveys (CSUR) 4*, 3 (1972), 179–196.

[60] Hopcroft, J. E.; Karp, R. M. An n^5/2 algorithm for maximum matchings in bipartite graphs. *SIAM Journal on computing 2*, 4 (1973), 225–231.

[61] Hunter, P.; Kreutzer, S. Digraph measures: Kelly decompositions, games, and orderings. *Theoretical Computer Science 399*, 3 (2008), 206 – 219. Graph Searching.

[62] Impagliazzo, R.; Paturi, R. Complexity of k-sat. In *Computational Complexity, 1999. Proceedings. Fourteenth Annual IEEE Conference on* (1999), IEEE, pp. 237–240.

[63] Impagliazzo, R.; Paturi, R.; Zane, F. Which problems have strongly exponential complexity? In *Foundations of Computer Science, 1998. Proceedings. 39th Annual Symposium on* (1998), IEEE, pp. 653–662.

[64] Impagliazzo, R.; Paturi, R.; Zane, F. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences 63*, 4 (2001), 512–530.

[65] Johnson, T.; Robertson, N.; Seymour, P.; Thomas, R. Directed tree-width. *Journal of Combinatorial Theory, Series B 82*, 1 (2001), 138 – 154.

[66] Jünger, M.; Reinelt, G.; Rinaldi, G. *Combinatorial Optimization–Eureka, You Shrink!: Papers Dedicated to Jack Edmonds. 5th International Workshop, Aussois, France, March 5-9, 2001, Revised Papers*, vol. 2570. Springer, 2003.

[67] Karp, R. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, R. Miller, J. Thatcher, and J. Bohlinger, Eds., The IBM Research Symposia Series. Springer US, 1972, pp. 85–103.

[68] Katrenic, J.; Semanisin, G. A generalization of hopcroft-karp algorithm for semi-matchings and covers in bipartite graphs. *arXiv preprint arXiv:1103.1091* (2011).

[69] Kshemkalyani, A. D.; Singhal, M. On characterization and correctness of distributed deadlock detection. *Journal of Parallel and Distributed Computing 22*, 1 (1994), 44–59.

[70] Kshemkalyani, A. D.; Singhal, M. *Distributed computing: principles, algorithms, and systems*. Cambridge University Press, 2011.

[71] Leung, JY-T.; Lai, E. K. On minimum cost recovery from system deadlock. *IEEE Transactions on Computers 9*, C-28 (1979), 671–677.

[72] Lokshtanov, D.; Marx, D.; Saurabh, S., et al. Lower bounds based on the exponential time hypothesis. *Bulletin of EATCS 3*, 105 (2013).

[73] Maheshwari, M. K.; Bansal, A. Process resource allocation in grid computing using priority scheduler. *International Journal of Computer Applications (0975–8887) 46*, 11 (2010), 20–23.

[74] Misra, J.; Chandy, K. M. A distributed graph algorithm: Knot detection. *ACM Transactions on Programming Languages and Systems (TOPLAS) 4*, 4 (1982), 678–686.

[75] Niedermeier, R. *Invitation to Fixed-Parameter Algorithms.* OUP Oxford, 2006.

[76] Oliveira, M. d. O. Subgraphs satisfying mso properties on z-topologically orderable digraphs. In *International Symposium on Parameterized and Exact Computation* (2013), Springer, pp. 123–136.

[77] Oliveira, M. d. O. An algorithmic metatheorem for directed treewidth. *Discrete Applied Mathematics 204* (2016), 49–76.

[78] Oum, S.-I. Approximating rank-width and clique-width quickly. *ACM Transactions on Algorithms 5*, 1 (2008), 10:1–10:20.

[79] Pachl, J. Deadlock avoidance in railroad operations simulations. In *Transportation Research Board 90th Annual Meeting* (11-0175, 2011).

[80] Pachl, J. The deadlock problem in automatic railway operation. *Signal und Draht. Vol. 89, no. 1-2* (1997).

[81] Penso, L. D.; Protti, F.; Rautenbach, D.; dos Santos Souza, U. Complexity analysis of P_3-convexity problems on bounded-degree and planar graphs. *Theoretical Computer Science 607* (2015), 83–95.

[82] Rabinovich, R.; Forschungsgebiet, L.-u. *Complexity measures of directed graphs.* Tese de Doutorado, RWTH Aachen University, 2008.

[83] Ryang, D.-S.; Park, K. H. A two-level distributed detection algorithm of AND/OR deadlocks. *Journal of Parallel and Distributed Computing 28*, 2 (1995), 149–161.

[84] Safari, M. A. D-width: A more natural measure for directed tree width. In *Mathematical Foundations of Computer Science 2005* (Berlin, Heidelberg, 2005), J. Jędrzejowicz and A. Szepietowski, Eds., Springer Berlin Heidelberg, pp. 745–756.

[85] Satyanarayana, B.; Prasad, K. S. *Discrete Mathematics and Graph Theory.* PHI Learning Pvt. Ltd., 2014.

[86] Singhal, M. Deadlock detection in distributed systems. *Computer 22*, 11 (1989), 37–48.

[87] Tanenbaum, A. S.; Woodhull, A. S. *Operating systems: Design and Implementation*, vol. 2. Prentice-Hall Englewood Cliffs, NJ, 1987.

[88] Terekhov, I.; Camp, T. Time efficient deadlock resolution algorithms. *Information Processing Letters 69*, 3 (1999), 149–154.