Approximation of Controllability Graphs via Power Dominating Set

Bader Alwasel

Thesis submitted to the University of London
for the degree of Doctor of Philosophy

2016

Approximation of Controllability Graphs via Power Dominating Set

School of Mathematics and Information Security
Royal Holloway, University of London

Declaration of Authorship

I, Bader Alwasel, hereby declare that this thesis and the work presented in it is entirely my own. Where I have consulted the work of others, this is always clearly stated. This work has not been submitted for any other degree or award in any other university or educational establishment.

Bader Alwasel
January, 2016

The starting point of all achievement is desire.

Napoleon Hill [1883-1970]

Acknowledgments

First and foremost, I am extremely grateful to Almighty God who has bestowed his countless blessings on me to accomplish my doctoral studies.

I would like to express my deepest gratitude to my supervisor, Dr. Stephen D. Wolthusen, for always giving me magnificent guidance, encouragement and support over these years. His versatile and profound knowledge provided me with great suggestions and inspiration in conducting my research. Besides learning academic skills from him, his personality, attitude, positive thinking and perception broadened my mind. Indeed, without his insightful ideas, invaluable comments and precious feedback, this thesis would never have become reality.
I would also like to thank my external examiner prof. Miguel Correia of University of Lisbon and my internal examiner Prof. Jason Crampton of Royal Holloway for their insightful views and comments on my thesis.

Furthermore, I want to thank my parents who have always been my inspiration. I have achieved everything only due to their prayers and efforts. I owe a lot to them for their constant encouragement and support to finish this long journey throughout the years of my studies. My gratitude extends also to my brothers and sisters who encouraged and supported me in every possible way to complete this work. I am very much indebted to my wife, Atheer, for her endless support, continued encouragement and great patience during my PhD studies. Finally, I would like to thank my government for sponsoring and supporting me.

Abstract

The concept of *controllability* was introduced by R. Kalman, which requires that a desired configuration can be forced from an arbitrary configuration in a finite number of steps. *Controllability* offers a comprehensive, rigorous and detailed framework for the design and analysis of not only control systems, but also of networks requiring a control relationship between vertices. The safe, secure, and effective operation of critical infrastructures such as electric power relies on the ability to monitor the state of a given system or network, or more formally the ability to observe the state and to estimate the state of a system. This is a pre-requisite for the ability to force a system from one state to another within a pre-defined finite interval, i.e. the problem of *Kalman controllability*, studied extensively in control theory. The problem of *structural controllability* originally defined by Lin [69] offers a graph theoretical interpretation for control systems as first described by Kalman, which is particularly suitable for studying sets of nodes offering the ability to control an entire system as represented by a control graph. The identification of minimum *Driver Nodes* (D_N) via the *Maximum Matching* was proposed by Liu *et al*. [71] as a powerful mechanism, offering full control over the network and an obvious target for attackers to disrupt these relations or compromise intermediate nodes, thereby gaining partial or total control of a distributed system.

Both attackers and defenders can hence identify nodes of particular interest, thereby strongly motivating the development of algorithms for identifying such sets of D_N, particularly after an attack or reconfiguration of the underlying network. This offers a strong motivation to study the ability of such systems to recover from deliberate attacks.

This thesis studies the alternative approach based on the POWER DOMINATING SET (PDS) problem, which gives an equivalent formulation for identifying minimum *Driver Nodes* (N_D). We also describe the problems of *controllability* and *structural controllability* as represented by the PDS problem and investigate different attacks affecting control networks. We therefore review existing work on graph classes, for which a PDS has been studied before, identifying a possible embedding of such structures in Erdős-Rényi graphs of different density as well as the approximation characteristics, which can be achieved in order to adapt them for solving the DIRECTED POWER DOMINATING SET problem. This allows the rapid identification of feasible alternative control structures where attackers have damaged or compromised the original control network, and the recovering of partial *controllability* if a control network has been partitioned.

We therefore propose a reconstruction algorithm for (directed) control graphs of bounded *tree-width* embedded in Erdős-Rényi random graphs based on recent work by Aazami and Stilp as well as Guo *et al*. This algorithm considers the rapid reconstruction of a PDS under attack as the more critical requirement relative to optimising the resulting PDS and hence propose an approximation based on a dynamic programming approach for directed graphs, where a tree of bounded width can be embedded in an Erdős-Rényi random graph.

We also study the case of sparse Erdős-Rényi graphs with directed control edges and seek to reduce the average-case complexity of a reconstruction algorithm for (directed) control graphs proposed in Chapter 4. We therefore obtain an enhanced average-case complexity of the recovery algorithm based on a DFS structure after an event or attack leading to disrupt legitimate control and compromise controllability of dependent nodes or disconnect parts of the control original graph. This DFS-based approach reduces the average-case complexity of the recovery algorithm by re-using remaining fragments of the original, efficient control graph where possible and identifying previously un-used edges to minimise the number of a PDS.

Furthermore, we study the *structural controllability* properties of the control graph in *Linear Time-Invariant* systems (LTI) via the PDS problem introduced by Haynes for studying power networks, addressing the question of how to recover a control graph as far as possible if the PDS or its dependent nodes have been partially compromised without complete re-computation. The approach is based on a BLOCK DECOMPOSITION of a directed graph, allowing us to identify *Cut-Vertices* (or a *articulation point*) and cut-edges. This results in faster re-construction of a minimal PDS structure, and ultimately the re-gaining of control for operators of control systems by applying three phases.

In addition, we study the case of sparse Erdős-Rényi graphs with directed control edges and seek to investigate the effect of rewiring edges on the *structural controllability* properties of directed Erdős-Rényi graphs in order to achieve a minimal PDS while keeping the total number of edges unchanged. The approach is based on a STAR DECOMPOSITION of a directed graph, allowing us to identify the number of *out-neighbours* of a PDS, and ultimately achieving of a minimal PDS.

Contents

List of Figures

List of Tables

List of Theorems

List of Lemmata

List of Definitions

List of Algorithms

Chapter 1

Introduction

1.1 Overview

This chapter gives an overview of the research questions in this thesis. We provide the motivation for the research and describe the contributions of this thesis. The overall structure of the thesis is hereby presented.

1.2 Motivation

Control systems are ubiquitous in cyber-physical systems as employed in most critical infrastructure systems. Large-scale distributed control systems such as those encountered in electric power networks or industrial control systems could be vulnerable to attacks, in which adversaries can take over control of at least part of the control network by compromising a subset of nodes. The problem of *controllability* of networks arises in different domains, including critical infrastructure systems that are increasingly vulnerable to a dangerous mix of traditional and nontraditional types of threats [35, 90]. The SCADA (Supervisory Control And Data Acquisition) environment faces a unique challenge to secure physically the network itself, as the nodes are scattered over a large geographical area. For instance, the Stuxnet attack on the Iranian uranium-processing equipment galvanised researchers and security professionals into focusing far more closely on the threats posed to the critical infrastructure industries [41].

Controllability theory offers a comprehensive, rigorous and detailed framework for the design and analysis of not only control systems but also of networks requiring a control relationship between vertices. *Controllability* informally the ability to

force a system into a desired state in a finite time or number of steps, is a fundamental problem studied extensively in control systems theory. In distributed control systems, possible control relations between vertices are limited by the underlying network (graph) transmitting the control signals from a single controller or set of controllers. Attackers may seek to disrupt these relations or compromise intermediate nodes, thereby gaining partial or total control.
For a defender to re-gain full or partial control, it is therefore critical to rapidly reconstruct the control graph as far as possible. Failing to achieve this may allow the attacker to cause further disruptions, and may as in the case of electric power networks also violate real-time constraints leading to catastrophic loss of control. This offers a strong motivation to study the ability of such systems to recover from deliberate attacks.

Recent work conducted by Liu *et al*. [71] has renewed interest in the seminal work by Lin [69] on *structural controllability*, which provides a graph-theoretical interpretation to Kalman algebraic criterion. This also allows the identification of necessary and sufficient conditions for the identification of individual *Driver Nodes* (D_N) able to control a system with a given structure (topology). The ability to identify *Driver Nodes* must be considered crucial for both attackers and defenders in control systems where *Driver Nodes* (D_N) offer an obvious target for attackers to disrupt the network control. There are several methods of identifying *Driver Nodes*, but most attention has been paid to the *Maximum Matching* approach [71]; this approach by Liu *et al*. is based on a non-rigorous variant of the *Maximum Matching* problem to identify a subset of Driver Nodes.

We study an alternative approach based on the POWER DOMINATING SET (PDS) problem originally proposed by Haynes *et al*. [52] as a refinement of DOMINATING SET. This approach gives an equivalent formulation for identifying minimum *Driver Nodes*. As electric power networks must be maintained continuously to monitor the state of system as defined by a set of state variables, one method of monitoring these variables is to place as few measurement devices, which measure the state

variables in these systems, as possible at some locations. Since the cost of these devices is rather high, the ability to minimise their numbers is highly desirable while monitoring the entire system [52]. However, the problem of locating the smallest set of sensors to monitor the entire system is a graph theory problem introduced by Haynes *et al*. as a model for studying *electric power networks*, and as an extension to the well-known DOMINATING SET (DS) problem. Haynes *et al*. [52] showed that a PDS is **NP**-complete even when restricted to bipartite graphs or chordal graphs. However, there are some dynamic programming algorithms proposed by Aazami and Stilp [1] as well as Guo *et al*. [49] that find approximation algorithms to solve the PDS problem optimally in polynomial time on graphs of bounded *tree-width*, where a PDS is only approximable with recent results by Aazami bounding this to a factor of $2^{\log^{1-e} n}$, unless $\mathbf{NP} \subseteq DTIME(n^{polylog(n)})$.

1.3 The Erdős-Rényi Model

Erdős-Rényi [40] published a seminal article in which they introduced the concept of a random graph as defined in the following:

Definition 1.1 (The Erdős-Rényi Model)

Given a positive integer n and a probability value $0 \leq p \leq 1$, define the graph $G(n,p)$ to be the undirected graph on n vertices whose edges have probability p of existing such that for all pairs of vertices v, w there is an edge (vw) with probability p, where the number of edges in a $G(n,p)$ graph is a random variable with expected value $\binom{n}{2}p$.

As a baseline, we initially study directed Erdős-Rényi graphs on the basis that:

a. They represent a widely studied class of graphs that has been extensively considered in respect of various problems concerning graph theory, and
b. random graphs constitute an important and active research area, with numerous models that have been applied to communication networks.

1.4 Research Questions

The problem of defending a distributed system against subversion and external attacks can be viewed in the light of recent studies in *controllability* theory. As *structural controllability* proposed by Lin [69] provides a graphical-theoretical interpretation for Kalman algebraic criterion, we represent the solutions of the research problems, set forth in this thesis, from a graphical-theoretical point of view and not from an engineering perspective. We therefore design algorithms that have the ability to regain or maintain approximate *structural controllability* for directed Erdős-Rényi random graphs via the PDS formulation in the case of removing the vertices or/and edges of a directed graph. The algorithms presented in this thesis do not focus on how the control graph has been attacked; rather, we assume that a given graph is subjected to intentional or random removal.

1.4.1 Preliminary Definitions and Notation

Erdős-Rényi (ER) is a model for generating random graphs. Let a graph $G = (V, E)$, generated by ER, represents a control network *e.g.* for a power network where a vertex represents an electrical or control node (a substation bus where transmission lines, loads, and generators are connected) and an edge may represent a transmission line or a communication link joining two electrical nodes. A vertex u is called an *out-neighbour* of a vertex v if there is a directed edge from v to u in G. Similarly, u is called an *in-neighbour* of v if the directed edge (uv) is present. The *neighbourhood* of a vertex v in the graph G, denoted $N_G(v)$, is the set $N_G(v) = u \in V : uv \in E$. The members of $N_G(v)$ are called the neighbours of v. The *closed neighbourhood* of a vertex v, denoted $N_G[v]$, is the set $N_G[v] = N_G(v) \cup v$. The degree of a vertex v is denoted by $d(v)$. The number of *out-neighbours* of v is called the out-degree of v denoted by $d^+(v)$, the in-degree $d^-(v)$ is defined similarly. A directed edge (vu) that points from vertex v to vertex u is said to be incident from v and incident to u. A path in G from a vertex u to a vertex v is a sequence of distinct vertices $u = v_0, v_1, \ldots, v_t = v$ so that (v_i, v_{i+1}), $i = 0, \ldots, t-1$, are in $E(G)$. A path from u

to v together with the edge (vu) is called a cycle. A directed graph is weakly connected if there is an undirected *path* between any pair of vertices in the underlying undirected graph, which it is obtained by ignoring the directions of the edges of the directed graph.

1.4.2 Assumptions

Throughout this thesis, the graph that we consider is a directed graph $G = (V, E)$, constructed as $ER(n, p)$ with a set of vertices n, where each edge included in the graph G is determined independently with the edge probability p such that each pair of vertices $u, v \in n$ is connected with the same edge probability for each direction. We rely on a number of assumptions:

1. For the resulting instances of $ER(n, p)$, we consider the directed graphs that have no self-loops nor parallel edges, but may have two edges with different directions on the same two end vertices (called antiparallel edges) and may have directed cycles.
2. We consider the directed connected graphs such that the resulting instance of $ER(n, p)$ is denoted by $G^{'} = G \setminus M$, where $M = (V_M, E_M)$ denotes a set of isolated vertices form G such that $\forall v \in V_M$, the out-degree of a vertex v is $d^{+}_{M}(v) = 0$ and the in-degree of a vertex v is $d^{-}_{M}(v) = 0$ (i.e. there is no directed in-edge from $u \in G$ to $v \in V_M$). Similarly, no directed out-edge from $v \in V_M$ to $u \in G$ such that $G^{'} = G \setminus M$ denotes a set of connected vertices V and a set of possible edges E.
3. $G^{'}$ is weakly connected, where the underlying undirected graph is connected, such that there is an undirected path from u to v and a directed path from v to u. Note that the implication of using weakly connected graphs is to avoid the trivial solutions when assuming strongly connected graphs to reconstruct a PDS.
4. The algorithms presented in this thesis assume a complete view of the status of the graph after an attack, and the computation time of the algorithms is related to time complexity and not real time.

1.4.3 Structural Controllability Analysis via Embedding Power Dominating Set Approximation in Erdős-Rényi Graphs

The first research question is to describe the problems of *controllability* and *structural controllability* as represented by the PDS problem and investigate different attacks affecting control networks. We therefore review existing work on graph classes, for which a PDS has been studied before, identifying a possible embedding of such structures in Erdős-Rényi graphs of different density as well as the approximation characteristics, which can be achieved in order to adapt the ideas used for solving the DIRECTED POWER DOMINATING SET problem. This allows the rapid identification of feasible alternative control structures, where attackers have damaged or compromised the original control network, and the recovering of partial *controllability* if a control network has been partitioned (see Chapter 3).

1.4.4 Reconstruction of Structural Controllability over Erdős-Rényi Graphs via Power Dominating Sets

For the second research question, let $G^{'} = (V, E)$ be a directed graph, reconstructed as $ER(n, p)$, and given an instance of a *Directed PDS*, defined by $S^{'}$, for $G^{'}$. Assume that attackers in a position to eliminate some vertices of $G^{'}$ (i.e. in real-world context when an electric actuator or industrial sensor in power systems are subjected to intentional or random removal). This deletion of vertices may lead to a disconnected component of a directed graph $G^{'}$, defined by $H = (V, E)$, where $H = (V, E) \subset G^{'}$ is partitioned from the original graph $G^{'}$ such that $V(H) \notin V(G^{'})$ and $E(H) \notin E(G^{'})$ (i.e. there exists no edge in H whose one end vertex is in $G^{'}$ and verse vice). As a result, a PDS of a directed graph $H = (V, E)$ (i.e. a disconnected component of graph $G^{'}$) may have a different PDS from the remainder of $S^{'}$. The question is how to regain or maintain structural control of H (i.e. how to recover a *Directed PDS* for a given directed graph $H = (V, E)$ in the presence of attackers in a position to eliminate vertices of the control graph). We therefore propose a reconstruction algorithm for (directed) control graphs of bounded *tree-width*

embedded in Erdős-Rényi random graphs based on recent work by Aazami and Stilp as well as Guo *et al*. This approach supposes that a *nice tree decomposition* of a directed graph $H = (V, E)$ is given such that the underlying undirected graph has bounded *tree-width* (see Chapter 4).

1.4.5 Recovering Structural Controllability on Erdős-Rényi Graphs via Partial Control Structure Re-Use

The contribution of this research question is to study the case of sparse Erdős-Rényi graphs with directed control edges and seek to reduce the average-case complexity of a reconstruction algorithm for (directed) control graphs proposed in the research question (1.4.4). While this does not improve the worst-case complexity, we obtain an enhanced average-case complexity that offers a substantial improvement where sufficient fragments of the original control graph remain, as would be the case where an adversary could only take over regions of the network and thereby achieve partial control. We therefore propose a novel algorithm based on a DFS structure, which yields an improved average-case complexity over previous research question (1.4.4), after an event or attack leading to disrupt legitimate control, and therefore, compromise controllability of dependent nodes or disconnect parts of the control original graph. This DFS-based approach reduces the average-case complexity of the recovery algorithm by re-using remaining fragments of the original, efficient control graph where possible and identifying previously un-used edges to minimise the number of a PDS (see Chapter 5).

1.4.6 Recovering Structural Controllability in the Presence of Compromised Nodes

Large-scale distributed control systems such as those encountered in electric power networks or industrial control systems could be vulnerable to attacks, in which adversaries take over control of at least part of the whole network by compromising a subset of nodes. Attackers may seek to disrupt these relations or compromise

intermediate nodes, thereby gaining partial or total control of a distributed system. The purpose of this research question is to investigate the *structural controllability* of the control graph in LTI systems, and address the question of how to recover a control graph as far as possible if the PDS or its dependent nodes have been partially violated without complete re-computation. We therefore study the case of sparse Erdős-Rényi graphs with directed control edges and seek to provide an approximation of an efficient reconstructed control graph. The approach is based on a BLOCK DECOMPOSITION of a directed graph, allowing the identification of its *Cut-Vertices* and cut-edges, and ultimately the re-gaining of a PDS for a graph (see Chapter 6).

1.4.7 The Effect of Rewiring Edges on Structural Controllability

Electric power networks must be maintained continuously to monitor their systems state by placing as few measurement devices as possible at strategic locations. Because of the high cost of these devices, the ability to minimise their numbers is highly desirable for monitoring the entire system. However, the problem of monitoring an electric power system by placing as few measurement devices in the system as possible is closely related to the well-known domination problem in graphs. We therefore study the case of sparse Erdős-Rényi graphs with directed control edges to achieve a minimal PDS without changing the total number of edges while maintaining the *structural controllability* of a graph. The approach is based on a DIRECTED STAR decomposition of a directed graph, allowing us to identify the number of *out-neighbours* of a PDS, and ultimately achieving of a minimal PDS (see Chapter 7).

1.5 Structure of the Thesis

This thesis is structured as follows: Chapter 2 introduces a literature review of *controllability* and *structural controllability* as represented by the PDS problem. Chapter 3 reviews existing work on graph classes, for which a PDS has been studied before, identifying a possible embedding of such structures in Erdős-Rényi

graphs of different density as well as the approximation characteristics, which can be achieved in order to adapt them for solving the DIRECTED POWER DOMINATING SET problem. Chapters 4 to 6 are previously published papers as shown in the following section.

1.6 Publications

The material of this thesis contains papers previously published in conjunction with my academic supervisor Dr. Stephen D. Wolthusen, as follows:

- **Chapter 3 [12]:** Analysis of the *structural controllability* of the control graph over directed Erdős-Rényi graphs via the POWER DOMINATING SET problem is undertaken in this paper reviewing existing work on graph classes for which a PDS has been studied before in order to adapt the methods used for solving the DIRECTED POWER DOMINATING SET problem.
- **Chapter 4 [13]:** This paper contributes to recovering the control graph via computing a *Directed PDS* for a given graph when adversaries have the ability to compromise *controllability* of dependent nodes or disconnect parts of the original control network.
- **Chapter 5 [14]:** We propose a novel algorithm using a DFS-based approach, which yields an improved average-case complexity of a reconstruction algorithm for (directed) control graphs over the previous work [13].
- **Chapter 6 [15]:** The contribution of the paper is to investigate the *structural controllability* of the control graph in LTI systems, and address the question of how to restore a control graph as far as possible in the presence of such *compromised nodes* without complete re-computation.
- **Chapter 7 [submitted to review]:** This paper studies the case of sparse Erdős-Rényi graphs with directed control edges to achieve a minimal PDS without changing the total number of edges while maintaining the *structural controllability* of a graph. The approach is based on a DIRECTED STAR decomposition of a directed graph, allowing us to identify the number of *out-neighbours* of a PDS, and ultimately achieving of a minimal PDS.

Chapter 2

Background

2.1 Overview

Controllability and *observability* represent two major concepts of modern control system theory. Most attention is being paid from various fields such as statistics, mathematics, computer science, biology, control theory and physics. Thus, extensive research has been proposed to show how control theory can be applied to network *controllability* [71, 73]. This chapter elaborates the problems of *controllability* and *structural controllability* as represented by the PDS problem with emphasis on *Linear Time Invariant* systems (LTI).

2.2 Controllability Theory

The concept of *controllability* was introduced by R. Kalman in 1960, which offers a comprehensive, rigorous and detailed framework for the design and analysis of not only control systems but also of networks requiring a control relationship between vertices. The safe, secure and effective operation of critical infrastructures such as electric power, telecommunications and computer networks relies on the ability to monitor the state of a given system or network, or more formally the ability to observe the state, and where this is not possible directly, to estimate the state of a system. This is a pre-requisite for the ability to force a system from one state to another within a pre-defined finite interval. The notion of control in network theory is described as the following: a directed network, N_2 is said to be controlled by another N_1, if there is a directed path from N_1 to N_2. However, the concept of *controllability* in control theory is defined as the behaviour of the network on the

basis of the dynamical model [73]. Informally, *controllability* requires that a desired configuration can be forced from an arbitrary configuration in a finite number of steps. More formally, *Kalman controllability* is defined (for the simple case of *a Time-dependent Linear Dynamical System*) as:

$$\dot{x}(t) = \mathbf{A}x(t) + \mathbf{B}u(t), \qquad x(t_0) \; = \; x_0 \tag{2.1}$$

with $x(t) = (x_1(t), \ldots, x_n(t))^T$ the current state of a system with n nodes at time t, a $n \times n$ adjacency matrix $\mathbf{A}$ representing the network topology of interactions among nodes, and $\mathbf{B}$ the $n \times m$ *input* matrix ($m \leq n$), identifying the set of nodes controlled by a time-dependent *input vector* $u(t) = (u_1(t), \ldots, u_m(t))$ which forces the desired state. The system in equation (2.1) is *controllable* if and only if:

$$\text{rank} \;\; [\mathbf{B}, \mathbf{AB}, \mathbf{A}^2\mathbf{B}, \ldots, \mathbf{A}^{n-1}\mathbf{B}] = n \qquad (\textit{Kalman rank criterion}) \tag{2.2}$$

giving the mathematical condition for *controllability*, where the rank of the *controllability* matrix provides the dimension of the controllable subspace of the system (A, B). However, it is computationally hard to verify this criterion for large complex networks, as the number of input combinations grows exponentially with the number of nodes ($\sim 2^N$) [71, 105]. Therefore, since verifying the condition is known to be prohibitively expensive, approximations are required particularly for larger graphs. Efficient ways to achieve *structural controllability* of LTI systems together with robustness have been extensively studied in recent years [82, 89, 91].

2.2.1 The Formulations of Controllability

Continuous and discrete time systems that are both Linear and Time Invariant play a central role in digital signal processing, communication engineering and control applications. Therefore, the following shows how *controllability* of discrete-time systems is given in terms of the *controllability* matrix. However, the continuous-time system is not discussed in this thesis.

2.2.1.1 Discrete Linear Time Invariant Systems

Consider a linear discrete time invariant control system defined by:

$$\dot{x}(k+1) = \mathbf{A}_d x(k) + \mathbf{B}_d u(k), \qquad x(0) \; = \; x_0 \tag{2.3}$$

The system *controllability* is defined as an ability to transfer the system from any initial state $x(0) = x_0$ to any desired final state $x(k_1) = x_f$ in a finite time. In order to find a control sequence $u(0), u(1), \ldots, u(n-1)$, such that $x(k) = x_f$, assuming that the input $u(k)$ is a scalar, i.e. the input matrix $\mathbf{B}_d$ is a vector denoted by $\mathbf{b}_d$. Thus, we have:

$$\dot{x}(k+1) = \mathbf{A}_d x(k) + \mathbf{b}_d u(k), \qquad x(0) \; = \; x_0 \tag{2.4}$$

Taking $k = 0, 1, 2, \ldots, n$ in (equation: 2.4), we obtain the following set of equations:

$$\left.\begin{aligned} & x(1) = \mathbf{A}_d x(0) + \mathbf{b}_d u(0) \\ & x(2) = \mathbf{A}_d x(1) + \mathbf{b}_d u(1) = \mathbf{A}_d^2 x(0) + \mathbf{A}_d \mathbf{b}_d u(0) + \mathbf{b}_d u(1) \\ & \vdots \\ & x(n) = \mathbf{A}_d^n x(0) + \mathbf{A}_d^{n-1} \mathbf{b}_d u(0) + \ldots + \mathbf{b}_d u(n-1) \end{aligned}\right\} \tag{2.5}$$

(equation: 2.5) can be written in a matrix form as:

$$x(n) - \mathbf{A}_d^n x(0) = \left[\mathbf{b}_d \vdots \mathbf{A}_d \mathbf{b}_d \vdots \ldots \vdots \mathbf{A}_d^{n-1} \mathbf{b}_d\right] \begin{bmatrix} u(n-1) \\ u(n-2) \\ \vdots \\ u(1) \\ u(0) \end{bmatrix} \tag{2.6}$$

Note that $[\mathbf{b}_d \vdots \mathbf{A}_d \mathbf{b}_d \vdots \ldots \vdots \mathbf{A}_d^{n-1} \mathbf{b}_d]$ is a square matrix. It is called the *controllability* matrix and denoted by C. If the *controllability* matrix C is nonsingular, (equation:

2.6) produces the unique solution for the input sequence where:

$$\begin{bmatrix} u(n-1) \\ u(n-2) \\ \vdots \\ u(1) \\ u(0) \end{bmatrix} = \mathbf{C}^{-1}(x(n) - \mathbf{A}_d^n x(0)) \tag{2.7}$$

Thus, for any $x(n) = x_f$, the expression in (equation: 2.6) determines the input sequence that transfers the initial state x_0 to the desired state x_f in n steps. In this case, it follows that the *controllability* condition is equivalent to a non-singularity of the *controllability* matrix $\mathbf{C}$.

In a general case, when the input $u(k)$ is a vector of dimension r, the repetition of the same procedure as in (equation: 2.3) - (equation: 2.5) leads to:

$$x(n) - \mathbf{A}_d^n x(0) = \left[\mathbf{B}_d \vdots \mathbf{A}_d\mathbf{B}_d \vdots \ldots \vdots \mathbf{A}_d^{n-1}\mathbf{B}_d\right] \cdot \begin{bmatrix} u(n-1) \\ u(n-2) \\ \vdots \\ u(0) \end{bmatrix} \tag{2.8}$$

Thus, the *controllability* matrix $\mathbf{C}$, in the general vector input case, defined by:

$$\mathbf{C}(\mathbf{A_d}, \mathbf{B_d}) = \left[\mathbf{B}_d \vdots \mathbf{A}_d\mathbf{B}_d \vdots \ldots \vdots \mathbf{A}_d^{n-1}\mathbf{B}_d\right] \tag{2.9}$$

is of dimension $(n \times r.n)$. The corresponding system of linear algebraic equations in $r.n$ unknowns for n r-dimensional vector components of $u(0), u(1), \ldots, u(n-1)$, given by:

$$\mathbf{C}^{n\times(n.r)} \begin{bmatrix} u(n-1) \\ u(n-2) \\ \vdots \\ u(1) \\ u(0) \end{bmatrix}^{(n.r)\times 1} = x(n) - \mathbf{A}_d^n x(0) = x_f - \mathbf{A}_d^n x(0) \tag{2.10}$$

This equation (2.10) will have a solution for any x_f if and only if the matrix $\mathbf{C}$ has a full rank, i.e. $\mathbf{C} = n$ as defined in (equation: 2.9).

Theorem 2.1 (Controllability of LTI Systems)

An linear discrete time system is controllable if and only if:

$$rank \quad [C] = n \tag{2.11}$$

2.3 Structural Controllability

As the Kalman rank criterion gives the mathematical condition for *controllability*, where the rank of the *controllability* matrix provides the dimension of the controllable subspace of the system $(\mathbf{A}, \mathbf{B})$, the verification of the condition is prohibitively expensive in particular for large complex networks [71, 105]. Therefore, the *structural controllability* of *LTI systems* has been studied extensively after *seminal work of Lin* [33, 69, 71, 83]. The key contributions of Lin provided a graph-theoretical interpretation for control systems as first described by Kalman [59], which is particularly suitable for studying sets of nodes offering the ability to control an entire system as represented by a control graph.

The concept of *structural controllability* was first introduced by Lin in 1970s [69] where the basic idea relies on the LTI system (equation: 2.12). The main result of the seminal work by Lin shows that the system $(\mathbf{A}, \mathbf{B})$ is structurally controllable if and only if the representation of a graph of $(\mathbf{A}, \mathbf{B})$ is spanned by a *cactus* [69]. The basic idea is that the set of all controllable pairs in the system (equation: 2.12) is open and dense in the space of all pairs $(\mathbf{A}, \mathbf{B})$ with standard metric [69], such that if a pair $(\mathbf{A}_0, \mathbf{B}_0)$ is not controllable, then for every $\epsilon > 0$, there exists a completely controllable pair $(\mathbf{A}, \mathbf{B})$ with $||\mathbf{A} - \mathbf{A}_0|| < \epsilon$ and $||\mathbf{B} - \mathbf{B}_0|| < \epsilon$ where $||.||$ denotes matrix norm [69]. This result reflects a physical point of view. Practically, for every pair $(\mathbf{A}, \mathbf{B})$, most of values of $\mathbf{A}$ and $\mathbf{B}$ are not known precisely except the entries, which are equal to zero [69]. Thus, Lin assumes that some entries of $\mathbf{A}$ and $\mathbf{B}$ are precisely zero, while all the entries are known approximately. So, the system $(\mathbf{A_0}, \mathbf{B_0})$ is said to be structurally controllable if and only if there exists a completely controllable $(\mathbf{A}, \mathbf{B})$ which has the same structure as $(\mathbf{A_0}, \mathbf{B_0})$. This means that both the pair $(\mathbf{A}, \mathbf{B})$ and another pair $(\tilde{\mathbf{A}}, \tilde{\mathbf{B}})$, of the same dimensions

are the same structure, provided every fixed (zero) entry of the matrix $(\mathbf{AB})$, the corresponding entry of the matrix $(\tilde{\mathbf{A}}, \tilde{\mathbf{B}})$ is a fixed (zero) and, at the same time, for every fixed (zero) entry of $(\tilde{\mathbf{A}}, \tilde{\mathbf{B}})$, the corresponding entry of $(\mathbf{AB})$ is also a fixed (zero) [69].

However, the matrices $\mathbf{A}$ and $\mathbf{B}$ in (equation: 2.12) are considered to be structured ones, i.e. their elements are either fixed zeros or independent free parameters. This reflects the fact that in reality the system parameters are often not known precisely except the zeros that mark the absence of connections between components of the system. Therefore, if no entry of $(\mathbf{A}, \mathbf{B})$ includes a fixed (zero), then the pair $(\mathbf{A}, \mathbf{B})$ is structurally controllable. Conversely, if there are some entries of $(\mathbf{A}, \mathbf{B})$ with a fixed (zero) then the pair may not be structurally controllable. So, the system $(\mathbf{A}, \mathbf{B})$ is said to be structurally controllable, if the matrices $\mathbf{A}$ and $\mathbf{B}$ are structured. This implies it is possible to fix the elements of $\mathbf{A}, \mathbf{B}$ (free parameters) to certain values in order to obtain the system $(\mathbf{A}, \mathbf{B})$ controllable, where the matrix rank $[C] = n$ [69, 71].

2.3.1 The Graph of Pair $(\mathbf{A}, \mathbf{B})$

The factors $\mathbf{A}$ and $\mathbf{B}$ in (equation: 2.12) are matrices, where $\mathbf{A} \in R^{n \times n}$ is an adjacency matrix giving the network topology identifying interaction among vertices, and $\mathbf{A} \in R^{n \times m}$ is the input matrix, where $m \leq n$, identifying the set of vertices controlled by the input vector $u(t)$. The *structural controllability* theorem gives the sufficient and necessary condition for a system to be structurally controllable through using graph-theoretical interpretations [69]. The whole system is defined by $(\mathbf{A}, \mathbf{B})$, that can be represented by a directed graph $G(\mathbf{A}, \mathbf{B}) = (V, E)$ as given in [71], where $V = V_{\mathbf{A}} \cup V_{\mathbf{B}}$ are the set of vertices and $E = E_{\mathbf{A}} \cup E_{\mathbf{B}}$ are the set of edges. In this representation, $V_{\mathbf{B}}$ comprises vertices able to inject control signals into the entire network, i.e. those constituting $u(t)$ in (equation: 2.12).

$$\dot{x}(t) = \mathbf{A}x(t) + \mathbf{B}u(t), \qquad x(t_0) \; = \; x_0 \tag{2.12}$$

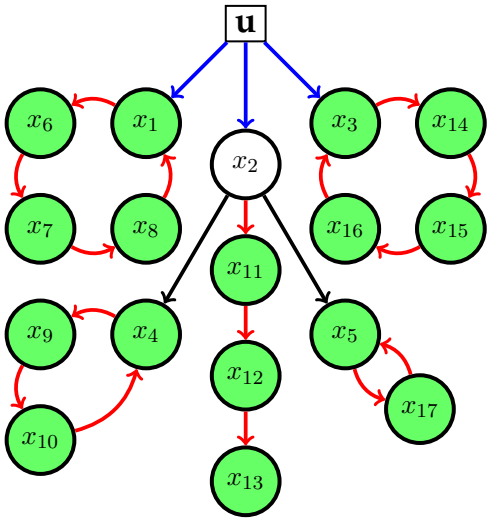
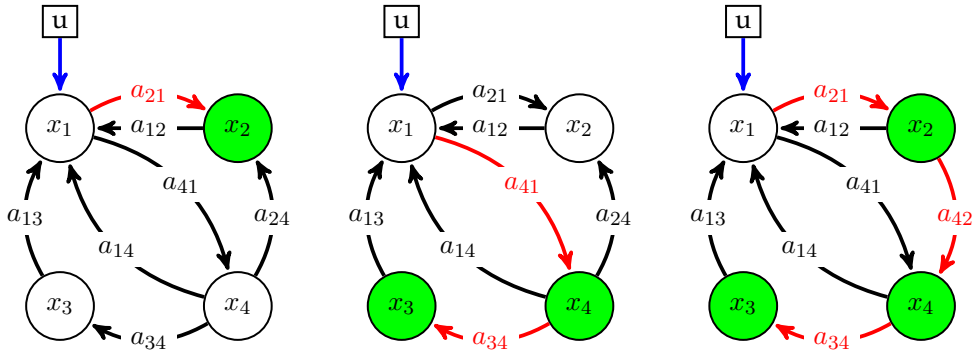


Figure 2.1: The Representation of the System (A,B) in Graph where the system is controlled by an input vertex (in the square form) and its edges are marked in blue. The green vertices denote matched nodes, and an unmatched node (marked in white) pointed by the input is called a controlled node (i.e. D_N). The red and black edges show *Matching* and non-*Matching* edges, respectively.

The system in (equation: 2.12) is denoted by $(\mathbf{A}, \mathbf{B})$, and it is represented by a directed graph $G(\mathbf{A}, \mathbf{B}) = (V, E)$ as given in [71], where $V = V_{\mathbf{A}} \cup V_{\mathbf{B}}$ are the vertex set and $E = E_{\mathbf{A}} \cup E_{\mathbf{B}}$ are the edge set. The set of state vertices is defined as $V_{\mathbf{A}} = x_1, \ldots, x_n := v_1, \ldots, v_n$, corresponding to the n nodes in the network. For instance, all vertices $\{x_1, \ldots, x_{17}\}$ in Figure 2.1 are called state vertices and the set of input vertices is defined as $V_{\mathbf{B}} = u_1, \ldots, u_m := v_{n+1}, \ldots, v_{n+m}$, corresponding to the m inputs (e.g. u in Figure 2.1). The set of edges between state vertices is defined as $E_{\mathbf{A}} = (x_j, x_i)|a_{ij} \neq 0$, i.e. the links in the network and the set of edges between input vertices and state vertices as $E_{\mathbf{B}} = (u_j, x_i)|b_{ij} \neq 0$. The m input vertices are also known as the origin of a directed graph $G(\mathbf{A}, \mathbf{B})$ and state vertices x_i connected to the origin u are called controlled nodes (e.g. x_1, x_2 and x_3 in Figure 2.1). Note one input vertex can be connected to multiple state vertices $V_{\mathbf{A}}$, where the number of controlled nodes $m^{'} \geq m$. the controlled nodes that do not share the input vertices u are defined by *Driver Nodes* (N_D) (e.g. x_2 in Figure 2.1). Thus, the number of *Driver Nodes* equals m, which is the number of inputs. So, if we control each node individually, i.e. $m = n$, then we are able to obtain full control for a system.



(a) Uncontrolled network (b) Uncontrolled network (c) Controlled network

Figure 2.2: An Example of Controllability, from [73], where the green and white vertices refer to matched and unmatched nodes, respectively. A square vertex implies an input vertex u with a blue edge, where the unmatched vertex pointed by the input vertex is called a controlled node (i.e. D_N). The red and black edges denote *Matching* and non-*Matching* edges, respectively.

The main goal of *controllability* is to control all nodes independently by a time-dependent input $u(t)$. For instance, Figure 2.2 represents the network with four nodes $\{x_1, x_2, x_3, x_4\}$ controlled by u, where the networks in Subfig 2.2.(a) and in Subfig 2.2.(b) are uncontrollable systems, because if the control flow goes through x_2, then the node x_4 can not be controlled (i.e. no edge points from x_2 to x_4) (see Subfig 2.2.(a)). On the other hand, if the *controllability* direction goes through x_4, then the nodes x_2 and x_3 are sharing the same superior x_4 owing to the violation of the *controllability* of a network (see Subfig 2.2.(b)). Hence, two or more subordinates should **not** share one superior in order to fully control a network (i.e. each node must be pointed by its own superior). However, a slight difference to the networks in Subfig 2.2.(a) and Subfig 2.2.(b) can make the networks controlled, where the edge between nodes x_1 and x_4 is reversed, hence the direction of *controllability* starts from $u \to x_1 \to x_2 \to x_4 \to x_3$, as in Subfig 2.2.(c).

2.3.2 Structural Controllability Theorem

The *structural controllability* of LTI systems has been well studied after the seminal work by Lin [69]. Before we state the theorem of Lin, we introduce several definitions.

Definition 2.1 (A Non-Accessible Node in *Structural Controllability*, Lin [69])
In a general graph, any vertex (v_i) *(except the origin nodes) is called* Non-Accessible *if and only if there are no directed paths reaching* v_i *from the origin node* v_{n+1}.

Definition 2.2 (*Dilation* in *Structural Controllability*, Lin [69])
Assuming that a set S *is formed by* k *nodes (other than the origin* v_{n+1}*) in the vertex set of a directed graph* $D(\mathbf{A}, \mathbf{B})$*, where* $S \subset V_{\mathbf{A}}$*. Whereby the set* $T(S)$ *of a set* S *is defined to be the set of all vertices* v_j *with the property that there exists an oriented edge from* v_j *to a vertex in* S*, where* $T(S) = v_j|(v_j \to vi) \in E(G), v_i \in S$*. A directed graph* $G(\mathbf{A}, \mathbf{B})$ *contains a* Dilation *if and only if there is a subset* $S \subset V_{\mathbf{A}}$ *such that* $|T(S)| < |S|$*, where* $|S|$ *or* $|T(S)|$ *is the cardinality of set* S *or* $T(S)$*, respectively.*

Definition 2.3 (Elementary Paths and Cycles in *Structural Controllability*, Liu [71])
For a directed graph, the set of directed edges $\{(v1 \to v_2), (v_2 \to v_3), \ldots, (v_{k-1} \to v_k)\}$ *where all vertices* $v_1, v_2, \ldots, v_k$ *are distinct is called an* elementary path *(*Stem*), and when* v_k *is incident to* v_1*, then it is named an elementary cycle (*Bud*).*

Definition 2.4 (A *Stem* in *Structural Controllability*, Lin [69])
A Stem *is an* elementary path *originating from an input vertex* $u_m \in V_{\mathbf{B}} = u_1, \ldots, u_m$*. The initial (or terminal) vertex of a* Stem *is called the root (or top) of the* Stem.

Definition 2.5 (A Bud in *Structural Controllability*, Lin [69])
A Bud *is an elementary cycle with an additional edge* e *that ends but not begins in a vertex of the cycle. The additional edge* e *is called the distinguished edge of the* Bud *such that the node* v_{n+1} *is called the origin of the* Bud *and the edge* $(v_{n+1} \to v_n)$ *is the distinguished edge of the* Bud.

Definition 2.6 (Cactus in *Structural Controllability*, Lin [69])
A Cactus *is a subgraph defined recursively as follows. A* Stem *is a* Cactus*. Given a* Stem S_0 *and buds* $\mathbf{B}_1, \mathbf{B}_2, \ldots, \mathbf{B}_l$*, then* $S_0 \cup \mathbf{B}_1 \cup \mathbf{B}_2 \cup \ldots \cup \mathbf{B}_l$ *is a* Cactus *if for every* i $(1 \leq i \leq l)$ *the initial vertex of the distinguished edge of* $\mathbf{B}_i$ *is not the top of* S_0 *and is the only vertex belonging at the same time to* $\mathbf{B}_i$ *and* $S_0 \cup \mathbf{B} \cup \mathbf{B}_2 \cup \ldots \cup \mathbf{B}_{i-1}$.

The combination of *Stems* and *Buds* is called a *Cactus* (defined below); therefore, if a graph of a pair $D(\mathbf{A}, \mathbf{B})$ is a *Stem* or a *Bud*, then the pair is structurally controllable (see Figure 2.3).

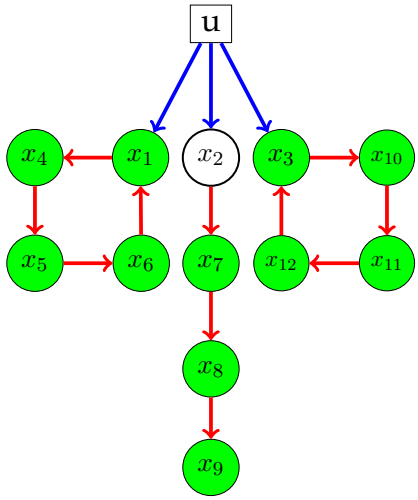


Figure 2.3: An Example of a *Cactus*, from [71]. This system, represented by a *Cactus*, is controlled by an input vertex u with blue edges. The green vertices denote matched nodes, and an unmatched node (marked in white) pointed by the input is called a controlled node (i.e. a *Driver Node*). The red and black edges represent *Matching* and non-*Matching* edges, respectively.

Example 1 Consider Figure 2.3, the *Cactus* contains a *Stem* $\{x_2, x_7, x_8, x_9\}$ and two *Buds* $\{x_1, x_4, x_5, x_6\}$ and $\{x_3, x_{10}, x_{11}, x_{12}\}$. Note that x_2 is a*Driver Node* where x_1 and x_3 are not sharing the same starting vertex u. □

Theorem 2.2 (*Structural Controllability*, Lin [69])

Any system $(\mathbf{A}, \mathbf{B})$ *is said to be structurally controllable if a linear control system* $(\mathbf{A}, \mathbf{B})$ *is structurally controllable, where a directed graph* $G(\mathbf{A}, \mathbf{B})$ *does not include any* Non-Accessible *node or* Dilation *such that the* $G(\mathbf{A}, \mathbf{B})$ *is spanned by* a Cactus.

2.3.3 Simple Examples of Controllability

The following examples, taken from [71], illustrate *structural controllability*:

Figure 2.4: Control a Simple Network (**1**). This network is controlled by an input vertex u with a blue edge. The green and white vertices denote matched and unmatched nodes, respectively. The vertex pointed by u is called a controlled node (i.e. a *Driver Node*). The red and black edges represent *Matching* and non-*Matching* edges, respectively.

Example (a)

The corresponding state-transition matrix in Figure 2.4 can be written as:

$$\begin{bmatrix} \dot{x_1}(t) \\ \dot{x_2}(t) \\ \dot{x_3}(t) \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ a_{21} & 0 & 0 \\ 0 & a_{32} & 0 \end{bmatrix} \cdot \begin{bmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \end{bmatrix} + \begin{bmatrix} b_1 \\ 0 \\ 0 \end{bmatrix} u(t) \tag{2.13}$$

The controllability *matrix is given by:*

$$\mathbf{C} = [\mathbf{B}, \mathbf{AB}, \mathbf{A}^2\mathbf{B}, \ldots, \mathbf{A}^{n-1}\mathbf{B}] = b_1 \begin{bmatrix} 1 & 0 & 0 \\ 0 & a_{21} & 0 \\ 0 & 0 & a_{32}a_{21} \end{bmatrix} \tag{2.14}$$

As the rank of the controllability *matrix (equation: 2.14) is* $\mathbf{C} = 3 = n$*, this system (equation: 2.13) is controllable where those weights (e.g.* a_{21}, a_{32} *and* b_1*) are non-zero, the system is always controllable. In other words, its* controllability *is independent of the detailed values of* a_{21}*,* a_{32} *and* b_1*.*

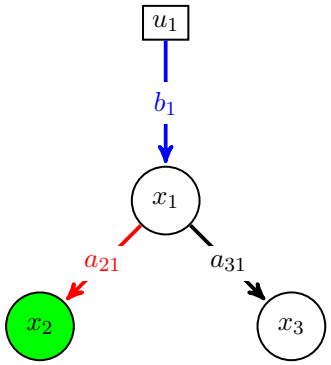


Figure 2.5: Control a Simple Network (**2**). This network is controlled by an input vertex u with a blue edge where the green and white vertices imply matched and unmatched nodes. The unmatched node pointed by the input is called a controlled node (i.e. a *Driver Node*). The red and black edges denote *Matching* and non-*Matching* edges, respectively.

Example (b)

The corresponding state-transition matrix in Figure 2.5 can be written as:

$$\begin{bmatrix} \dot{x_1}(t) \\ \dot{x_2}(t) \\ \dot{x_3}(t) \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ a_{21} & 0 & 0 \\ a_{31} & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \end{bmatrix} + \begin{bmatrix} b_1 \\ 0 \\ 0 \end{bmatrix} u(t) \tag{2.15}$$

The controllability *matrix is given by:*

$$\mathbf{C} = [\mathbf{B}, \mathbf{AB}, \mathbf{A}^2\mathbf{B}, \ldots, \mathbf{A}^{n-1}\mathbf{B}] = b_1 \begin{bmatrix} 1 & 0 & 0 \\ 0 & a_{21} & 0 \\ 0 & a_{31} & 0 \end{bmatrix} \tag{2.16}$$

This system in (equation: 2.15) is uncontrollable because the rank of the controllability *matrix (equation: 2.16) is* $\mathbf{C} = 2 < n$. *Although, the* controllability *of the detailed values of* a_{21}, a_{31}, *and* b_1 *are independent and non-zero, this system is uncontrollable, as it contains a* Dilation *in the place* $a_{31}x_2(t) = a_{21}x_3(t)$ *in the state space (i.e. fixed in* $a_{31}x_2(t) = a_{21}x_3(t)$*).*

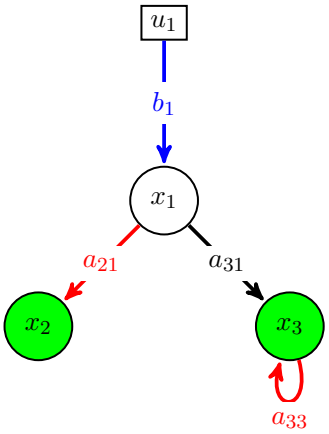


Figure 2.6: Control a Simple Network (**3**). This network is controlled by an input vertex u with a blue edge. The matched nodes are marked in green vertices, and an unmatched node (marked in white) pointed by the input is called a controlled node (i.e. a *Driver Node*). The red and black edges show *Matching* and non-*Matching* edges, respectively.

Example (c)

The corresponding state-transition matrix in Figure 2.6 can be written as:

$$\begin{bmatrix} \dot{x_1}(t) \\ \dot{x_2}(t) \\ \dot{x_3}(t) \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ a_{21} & 0 & 0 \\ a_{31} & 0 & a_{33} \end{bmatrix} \cdot \begin{bmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \end{bmatrix} + \begin{bmatrix} b_1 \\ 0 \\ 0 \end{bmatrix} u(t) \tag{2.17}$$

The controllability *matrix is given by:*

$$\mathbf{C} = [\mathbf{B}, \mathbf{AB}, \mathbf{A}^2\mathbf{B}, \ldots, \mathbf{A}^{n-1}\mathbf{B}] = b_1 \begin{bmatrix} 1 & 0 & 0 \\ 0 & a_{21} & 0 \\ 0 & a_{31} & a_{33}a_{31} \end{bmatrix} \tag{2.18}$$

As the rank of the controllability *matrix (equation: 2.18) is* $\mathbf{C} = 3 = n$*, this system (equation: 2.17) is controllable. However, this example is similar to the example in Figure 2.5 with a slight difference, which is the presence of a self-edge. The alteration makes this system is controllable. Note that as long as they are non-zero, the* controllability *of the detailed values of* a_{21}*,* a_{31} *,* a_{33}*, and* b_1 *are independent.*

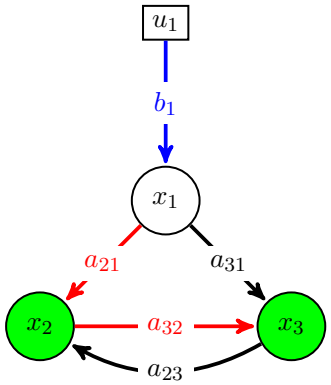


Figure 2.7: Control a Simple Network (**4**). This network is controlled by an input vertex u with a blue edge where the green vertices denote matched nodes. An unmatched node pointed by u is called a controlled node (i.e. a *Driver Node*). The red and black edges represent *Matching* and non-*Matching* edges, respectively.

Example (d)

The corresponding state-transition matrix in Figure 2.7 can be written as:

$$\begin{bmatrix} \dot{x_1}(t) \\ \dot{x_2}(t) \\ \dot{x_3}(t) \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ a_{21} & 0 & a_{23} \\ a_{31} & a_{32} & 0 \end{bmatrix} \cdot \begin{bmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \end{bmatrix} + \begin{bmatrix} b_1 \\ 0 \\ 0 \end{bmatrix} u(t) \tag{2.19}$$

The controllability *matrix is given by:*

$$\mathbf{C} = [\mathbf{B}, \mathbf{AB}, \mathbf{A}^2\mathbf{B}, \ldots, \mathbf{A}^{n-1}\mathbf{B}] = b_1 \begin{bmatrix} 1 & 0 & 0 \\ 0 & a_{21} & a_{23}a_{31} \\ 0 & a_{31} & a_{32}a_{21} \end{bmatrix} \tag{2.20}$$

This system (equation: 2.19) is controllable, because the rank of the controllability *matrix (2.20) is* $\mathbf{C} = 3 = n$. *This system will be uncontrollable, in case of pathological situations, where* $\binom{a_{21}}{a_{31}} \propto \binom{a_{23}a_{31}}{a_{32}a_{21}}$ *(e.g.* $a_{32}a_{21}^2 = a_{23}a_{31}^2$*), therefore rank* $(\mathbf{C}) = 2 < n$. *However, the system (equation: 2.19) can be controllable, if it should change the weight of link . As a result, the system (equation: 2.19) can be structurally controllable as it is controllable for almost all combinations of weights.*

2.4 Minimum Inputs Theorem

The ability to efficiently identify *Driver Nodes* (N_D) results in certain assumptions as identified recently by Liu *et al*. [71] who implicitly gives a criterion for both attackers and defenders for vertices and edges to target. The identification of minimum *Driver Nodes* (N_D) via the *Maximum Matching* was proposed by Liu *et al*. [71] as a powerful mechanism offering full control over the network and an obvious target for attackers to disrupt these relations or compromise intermediate nodes, thereby gaining partial or total control of a distributed system.

Two main approaches have been studied for determining $V_{\mathbf{B}}$ (i.e. minimum (N_D)); most attention has been paid to the *Maximum Matching* approach [71] where the *Maximum Matching* in a directed network is computed by mapping it to a bipartite graph [71]. The contribution of Liu *et al*. enables us to find the size of *Maximum Matching* in the corresponding directed graph $G(\mathbf{A})$ in a large number of the different classes of random directed graphs by assuming that the nodes are not dynamic.

2.4.1 Complexity of The Maximum Matching

According to Liu *et al*. in order to control a system, it should first identify the set of nodes that, if driven by different signals, can offer full control over the network (i.e. the set of *Driver Nodes*).

$$\text{rank } [\mathbf{B}, \mathbf{AB}, \mathbf{A}^2\mathbf{B}, \ldots, \mathbf{A}^{n-1}\mathbf{B}] = n \qquad (\textit{Kalman rank criterion})$$

To compute *Kalman rank criterion* in an arbitrary network, it is important to know the weight of each link which are either unknown for most real networks or are known only approximately and are time dependent (for example Internet traffic) [71]. However, even if all weights are known, a brute-force search is required to compute the rank of $[C]$ for $(2^N - 1)$ distinct combinations that is a prohibitively expensive for large complex networks [71]. Therefore, to avoid the need to measure the link weights, Liu *et al*. note that the system $(\mathbf{A}, \mathbf{B})$ is *structurally controllable* [69] if it is possible to choose the non-zero weights in $\mathbf{A}$ and $\mathbf{B}$ such that the system

satisfies the rank of $\mathbf{C} = \mathbf{N}$. A structurally controllable system can be shown to be controllable for almost all weight combinations, except for some pathological cases with a zero measure that occur when the system parameters satisfy certain accidental constraints [69, 96]. Thus, *structural controllability* helps to overcome inherently incomplete knowledge of the link weights in $\mathbf{A}$. So, Liu *et al*. [71] proved that the minimum number of *Driver Nodes* needed to maintain full control of the network is determined by the *Maximum Matching* in the network as defined below.

2.4.2 The Maximum Matching

Liu *et al*. developed analytical tools to study the *controllability* of an arbitrary complex directed network, identifying the set of D_N with time-dependent control that can guide the entire dynamics of the system. The number of D_N is determined mainly by the degree distribution allowing to calculate the analytical result by using the cavity method developed in statistical physics to predict D_N from $P(k_{in}, k_{out})$ analytically, where $P(k_{in}, k_{out})$ denotes the number of incoming and outgoing links. Now we introduce basic definitions before giving the *Minimum Inputs* theorem:

Definition 2.7 (The Maximum Matching in a Directed Graph)

The Maximum Matching *in a directed graph is the maximum set of edges that do not share starting or ending nodes such that a vertex is matched if it is an ending vertex of an edge in the* Matching. *Otherwise, it is unmatched. It is called perfect if all vertices are matched.*

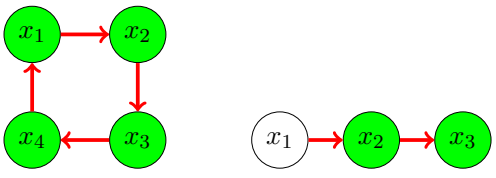


Figure 2.8: An example of the *Maximum Matching*. The green and white vertices denote matched and unmatched nodes, respectively. The red edges show *Matching* edges.

Note that the sufficient conditions to gain full control over a directed network as stated in [71] are to directly control each unmatched node and there are directed paths from the input signals to all matched nodes.

Example 2 Subfig 2.8.(a) is a *perfect matching* in G(**A**) where any node can be chosen as a *Driver Node*, while in Subfig 2.8.(b) the unmatched node (x_1) is a *Driver Node*.□

In general, there may be different *Maximum Matching* for a given graph or digraph. Therefore, the *Maximum Matching* of a directed graph $G(\mathbf{A})$ can be found by mapping it to a bipartite representation where vertices can be divided into two disjoint sets V and U such that every edge connects a vertex in V to one in U (see Figure 2.9).

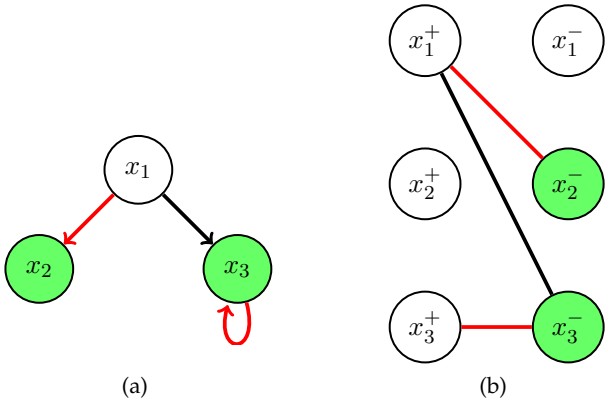


Figure 2.9: The *Matching* in a directed graph and its bipartite representation. The green and white vertices denote matched and unmatched nodes, respectively. In Subfig (b), the bipartite representation of the digraph shown in a simple digraph (a) has a unique *Maximum Matching*, which is shown in red edges. The black edges represent non-*Matching* edges.

The bipartite graph is defined in the following way: $H(\mathbf{A}) = (V_{\mathbf{A}}^+ \cup V_{\mathbf{A}}^-, \Gamma)$, where $V_{\mathbf{A}}^+ = (x_1^+, \ldots, x_n^+)$ and $V_{\mathbf{A}}^- = (x_1^-, \ldots, x_n^-)$ are the set of vertices corresponding to the n columns and rows of the State Matrix $\mathbf{A}$, respectively. Edge set $\Gamma = (x_j^+, x_i^-)|a_{ij} \neq 0$, such that any node i in a directed network of N nodes can have two separate components i^+ and i^-. So, the number of nodes in the bipartite representation is equal to $2n$ nodes [71]. If there is a directed edge from i to j in the original network, then there will be a directed edge from i^+ to j^- in the bipartite representation such that the *Maximum Matching* of bipartite representation is the same as that of the directed network.

Theorem 2.3 (Minimum Inputs, Liu *et al*. [71])

The minimum number of inputs (N_I) *or equivalently the minimum number of* Driver Nodes *(*N_D*) needs to fully control a network* $G(\boldsymbol{A})$ *is one if there is a* perfect matching *in* $G(\boldsymbol{A})$*, where any single node can be chosen as* N_D*. Otherwise, it equals the number of unmatched nodes with respect to any* Maximum Matching*, where* N_D *is just the unmatched nodes.*

2.4.2.1 The Maximum Matching Algorithm

The *Maximum Matching* of a bipartite network can be identified by some standard algorithms such as the Hungarian algorithm [64] and the Hopcrof-Karp algorithm [79]. The basic idea underlying both algorithms is based on finding *augmenting paths* that start at a free node, end at a free node, and alternate between unmatched and matched edges on the path, where a free node is not the ending vertex of an edge in some partial *Matching*. [105]. Now we introduce basic definitions [11]:

Definition 2.8 (Alternating Paths)

A Path P *is said to be an* alternating path *with respect to a* Matching M*, if and only if among every two consecutive edges along the path, only one belongs to* M*.*

Definition 2.9 (Augmenting Paths)

Given a graph $G = (V, E)$ *and a* Matching $M \subseteq E$*, a path* P *is called an* augmenting path *for* M *if:*

a. The two end points of P *are unmatched by* M*.*

b. The edges of P *alternate between edges* $\in M$ *and edges* $\notin M$*.*

In other words, an augmenting path *is an* alternating path *that starts from and ends in unmatched vertices.*

The main part of the Hungarian algorithm is to find a single *augmenting path* per iteration and it runs in time $\mathcal{O}(N^3)$ while the Hopcrof-Karp algorithm can produce a maximal set of the shortest *augmenting paths* per iteration, (i.e. the Hopcrof-Karp algorithm produces a set of as many edges as possible with the property that no two edges share a common vertex).

Definition 2.10 (Properties of Augmenting Paths)

1. *The number of links in any* augmenting path *is odd.*
2. *The* Matching *number of an* augmenting path *can be increased by 1 if unmatched and matched links are reversed.*
3. *A* Matching *is the* Maximum Matching *if and only if there is no* augmenting path *with respect to the* Matching.

Algorithm 2.1: Finding The *Maximum Matching* Set in a Bipartite Graph [55].

Input : A bipartite graph $G = (U \cup V, E)$.
Output: Matching $M \subseteq E$.

$M \leftarrow \emptyset$;
repeat ;
$P \leftarrow \{P_1, P_2, \ldots, P_k\}$ a maximal set of vertex-disjoint shortest augmenting paths ;
$M \leftarrow M \oplus (P_1 \cup P_2 \cup \ldots \cup P_k)$;
until $P = \emptyset$;
return M ;

Definition 2.11 (Hungarian Algorithm)

a. Initially, set the Maximum Matching *to be empty.*

b. Find an augmenting path *and then reverse all matched and unmatched links to obtain a larger* Matching.

c. Repeat step 2 until no more augmenting paths *can be found.*

2.5 Dominating Set (*DS*)

One of the major themes in graph theory is domination, and it is well known in graph theory as the DOMINATING SET (DS) problem (or vertex covering) for a graph [49, 52]. The basic problem is defined in the following definition:

Definition 2.12 (A *Dominating Set*)

A set $S \subseteq V$ is a Dominating Set *in a graph $G = (V, E)$ if every vertex in $V \setminus S$ has at least one neighbour in S, that is, $N_G[S] = V$.*

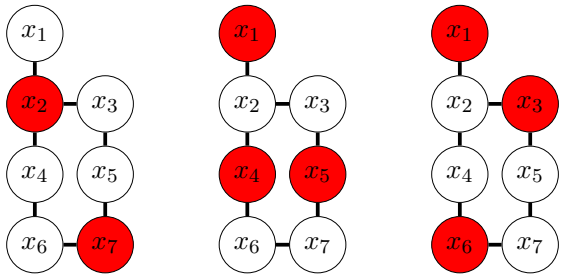


Figure 2.10: An example of a *Dominating Set* where the red vertices form a *Dominating Set* while the white vertices are the neighbours of a vertex in a *Dominating Set*

Consider Figure 2.10, a *Dominating Set* in each example can be different, implying DS is not unique. Given a graph $G = (V, E)$, where V denotes a set of vertices and E shows edges, determine a minimum vertex set $D \subseteq V$ such that every vertex $v \in V$ is contained in D or has at least one neighbour in D, i.e. every vertex not in D is adjacent to at least one member of D [52]. The problem of finding a minimum cardinality of a *Dominating Set* is a significant problem that has been extensively studied [1, 49, 52]. The minimum cardinality of a *Dominating Set* of G, denoted by $\gamma(G)$, is the number of vertices in the smallest *Dominating Set* for G. The basic decision problem DOMINATING SET is **NP**-complete and the parameterised intractability results imply $W[2]$-completeness [49]. Unless **NP**-hard problems have slightly super-polynomial time algorithms, *DS* is not polynomial time approximately better than $\Theta(\log |V|)$ [42, 49].

2.6 Power Dominating Set (PDS)

Electric power networks must be maintained continuously to monitor the state of system as defined by a set of state variables. One method of monitoring these variables is to place as few measurement devices as possible at some locations to measure the state variables in electric power systems. Since the cost of these devices is rather high, the ability to minimise their numbers is highly desirable while monitoring the entire system [52]. However, the problem of locating the smallest set of measurement devices is a graph theory problem introduced by Haynes *et al*. as a model for studying electric power networks and as an extension to the well-known DOMINATING SET problem, which is one of the classic decision problems [52, 63, 107].

Non-trivial control systems and controlled networks are necessarily sparse, and direct control of all nodes in such a network is not feasible as direct edges to these would typically result in exorbitantly high costs as well as an out-degree of the controller node that would be difficult to realise in larger networks. Instead, the general case to be considered is for control to be indirect. As control systems will seek to minimise parameters such as latency, the formulation for a PDS by Haynes *et al*. [52] extended the classic DOMINATING SET problem to obtain a minimal *Power Dominating Set*. The Power Domination in graphs can be summarised by two (simplified) *Observation Rules* (OR) due to Kneis *et al*. [63]:

OR1 A vertex in the PDS observes itself and all of its neighbours.

OR2 If an observed vertex v of degree $d \geq 2$ is adjacent to $d - 1$ observed vertices, then the remaining unobserved neighbour becomes observed as well.

Now, we define *Power Dominating Set* for a graph as follows:

Definition 2.13 (A Power Dominating Set for a Graph)

Input: *An undirected graph $G = (V, E)$ and an integer $k \geq 0$.*

Question: *Is there a set $P \subseteq V$ with $|P| \leq k$ which observes all vertices in V with respect to the two observation rules **OR1** and **OR2**?*

where P is called a PDS of G. Note the classical DOMINATING SET problem can be defined by simply omitting **OR2**

Definition 2.14 (A Power Dominating Set Problem)

Given an undirected graph $G = (V, E)$, find a minimum-size set $P \subseteq V$ such that all vertices in V are observed by the vertices in P

Here we consider a straightforward extension to directed graphs:

Definition 2.15 (A Directed Power Dominating Set Problem)

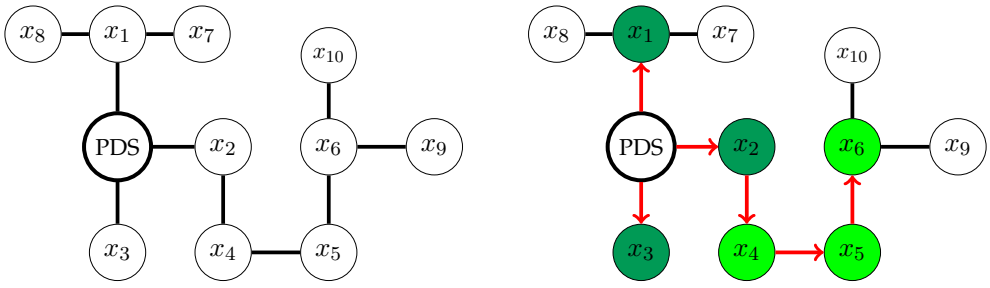
Given a directed graph $G = (V, E)$, find a minimum-size set $P \subseteq V$ such that all vertices in V are observed by the vertices in P

Definition 2.16 (A Directed PDS)

Let G be a directed graph. Given a set of vertices $S \subseteq V(G)$, the set of vertices power-dominated by S, denoted by $P(S)$, is obtained as:

D1 *If a vertex v is in S then v and all of its* out-neighbours *are in $P(S)$;*

D2 *If a vertex v is in $P(S)$, one of its* out-neighbours *denoted by w is not in $P(S)$ and all other* out-neighbours *of v are in $P(S)$ then w is inserted into $P(S)$. This condition is called a Propagation rule.*



(a) Before modeling the rules of Directed PDS (**D1**) and (**D2**) as stated in definition (2.16)

(b) After applying the rules of Directed PDS (**D1**), denoted by (dark-green) and (**D2**), shown by (light-green)

Figure 2.11: An example of Power Dominating Set

Example 3 Consider Figure 2.11, Subfig 2.11.(a) shows a graph before applying the first rule **[D1]** and Subfig 2.11.(b) represents a PDS obtained via applying the second rule **[D2]** where the set $\{x_1, x_2, x_3\}$ is covered by **[D1]** and the set $\{x_4, x_5, x_6\}$ is

power dominated by **[D2]**. However, the remaining vertices are still uncovered, as each of $\{x_1, x_6\}$ has two vertices that are not covered yet, meaning $\{x_1, x_6\}$ should be in a PDS. □

Therefore, the POWER DOMINATING SET problem is defined by the following question: Given a graph $G = (V, E)$ and set of vertices $P \subseteq V$, how to construct a set of vertices $C = V(G)$ that can be observed from P and a set of edges $F = E(G)$ are observed by P and also to minimise the size $|P|$. The following algorithm by Haynes *et al*. determines the sets of (observed) vertices C and edges F [52]:

Algorithm 2.2: The Construction of a PDS by Haynes *et al*.

Input : Given a graph $G = (V, E)$ and set of vertices $P \subseteq V$
Output: Construct a set of vertices $C = V(G)$ and edges $F = E(G)$ observed by P

1 Initialise $C = P$ and $F = \{e \in E(G) |\ e\ is\ incident\ to\ a\ vertex\ in\ P\}$;
2 Add to C any vertex not already in C which is incident to an edge in F;
3 Add to F any edge e not already in F which satisfies one of the following conditions: ;
4 **(a).** both end-vertices of e are in C;
5 **(b).** e is incident to a vertex v of degree greater than one, for which all the other edges incident to v are already in F;
6 If steps **2** and **3** fail to locate any new edges or vertices for inclusion, stop. Otherwise, go to step **2**. The final state of the sets C and F give the set of vertices and edges observed by the set P;

Chapter 3

PDS Algorithms-Related Work

3.1 Overview

Following the description of the problems of *controllability* and *structural controllability* as represented by the PDS problem; this chapter reviews existing work on graph classes, whereby a PDS has been studied prior in order to identify a potential embedding of such structures in Erdős-Rényi graphs for varied density and approximation characteristics, which may be realised for the purposes of making amendments to solve the DIRECTED PDS problem. This also allows the rapid identification of feasible alternative control structures where attackers have damaged or compromised the original control network, and to recover partial *controllability* if a control network has been partitioned.

3.2 NP-Completeness and Upper Bounds Results of Related Work

One problem immediately arising from edge/vertex removal is the *reconstruction* and recovery of control while obtaining a minimal PDS. In computational complexity theory, **NP**-completeness is described as the complexity class of decision problems, where **NP** stands for *"Non-deterministic Polynomial time"*. Informally, no polynomial solution to any of the **NP**-complete problems is available despite decades of intensive research; however, it is solvable in polynomial time by a non-deterministic turing machine [30].

The earliest publications on the PDS problem were by Brueni [28], Baldwin *et al*. [18], and Mili *et al*. [80]. However, the study of the PDS problem in respect with the graph theoretical representation was initiated by Haynes *et al*. [52] where they proved that a PDS for a given graph G is **NP**-complete for general graphs even when reduced to certain classes of graphs, such as bipartite graphs and chordal graphs, and they proposed a linear time algorithm for the PDS problem in trees. Guo *et al*. [49] showed that the PDS problem is also **NP**-complete for planar graphs, circle graphs, and split graphs, and it cannot be better approximated than the *DOMINATION* problem for general graphs. Parameterised results were given in [63, 49] and they proved $W[2]$-complete-hardness if the parameter is the size of the solution by reducing a *Dominating Set* to a PDS. Additionally, Guo *et al*. [49] showed fixed-parameter tractability of a PDS with respect to a tree decomposition of bounded *tree-width* for the underlying graph and provided a concrete algorithm transforming a PDS into an orientation problem on undirected graphs. For the DIRECTED PDS problem, Aazami and Stilp [1] presented a linear time dynamic programming algorithm when the underlying undirected graph has bounded *tree-width*.

As the PDS problem is a generalisation of the *Dominating Set* (DS) problem, the basic minimum DS problem is known to be **NP**-complete with a polynomial-time approximation factor of $\theta(\log n)$ as shown by Feige [42]. Subsequently, Aazami and Stilp [1] separated the approximation hardness of DOMINATING SET and PDS problems. They proved that the PDS problem cannot be approximated better than $2^{\log^{1-e} n}$, unless $\mathbf{NP} \subseteq DTIME(n^{polylog(n)})$. In addition, they proposed an $O(\sqrt{n})$-approximation algorithm for the PDS problem in planar graphs. Liao and Lee [68] showed a different **NP**-completeness proof for the PDS problem in split graphs, and they also presented a polynomial time algorithm for solving a PDS optimally on interval graphs. The PDS problem remains **NP**-hard on cubic graphs as proved by Binkele-Raible and Fernau [24]. Furthermore, Guo *et al*. [49] proposed valid orientations for optimally solving a PDS (over undirected graphs) on graphs of

bounded *tree-width*. Subsequently, Aazami and Stilp [2] reformulated the DIRECTED PDS (DPDS) as *Valid Colourings* of edges and proposed a Dynamic Programming algorithm for DPDS where the underlying undirected graph has bounded *tree-width*.

Graph Classes	**Dominating Set**	**Power Dominating Set**
Bipartite	**NP**-complete	**NP**-complete
Chordal	**NP**-complete	**NP**-complete
Circle	**NP**-complete	**NP**-complete
Comparability	**NP**-complete	**NP**-complete
Planar	**NP**-complete	**NP**-complete
Split	**NP**-complete	**NP**-complete
AT-free	poly. time	
Co-comparability	poly. time	
Distance hereditary	poly. time	
Dually chordal	Linear time	
Interval	poly. time	poly. time
k-polygon ($k \geq 3$)	poly. time	
Partial k-tree ($k \geq 1$)	Linear time	Linear time
Permutation	Linear time	
Strongly chordal	poly. time	
Block	Linear time	Linear time
Cubic	**NP**-hard	**NP**-hard
Series-parallel	Linear time	
Circular-arc	Linear time	Linear time

Table 3.1: The first part of the table is taken from [49] while the second part refers to some recent results on solving the PDS problem where empty entries imply that this has not been studied yet.

Some special classes of graphs have also been considered from an algorithmic point of view as shown in Table 3.1. The PDS problem in block graphs has been studied in [16, 107] where they proposed linear time algorithms for a PDS. While Hon *et al*. [54] proposed a linear time algorithm for the PDS problem on block-cactus graphs. Dorfling and Henning [38] determined the power domination number in grid graphs. Pai *et al*. [87] proposed a simpler algorithm for solving a PDS in grid graphs, relying on earlier results of Dorfling and Henning [38]. Dorbec *et al*. [37]

determined the power domination number for all direct products of paths except for the odd component of the direct product of two odd paths. Kao *et al*. [60] studied the PDS problem on honeycomb meshes and provided an algorithm to obtain a minimum PDS. Liao and Lee [67] proposed a linear time algorithm to solve a PDS in circular-arc graphs. Dean *et al*. [34] studied the *Power Domination* problem for a hypercube.

3.2.1 Upper Bounds Results of a PDS

Several studies investigated upper bounds for the *Power Domination* number on the different classes of graphs as shown in Table 3.2. In [109], Zhao *et al*. achieved an upper bound for a connected claw-free cubic graph. Xu *et al*. [107] provided a sharp upper bound for *Power Domination* number in block graphs and characterised the extremal graphs. In [106] the upper bounds of the power domination number are given for the generalised Petersen graphs, presenting both upper bounds for such graphs and exact results for a subfamily of generalised Petersen graphs. While Barrera and Ferrero [20] found upper bounds for the power domination number of some families of Cartesian products of graphs, the cylinders and the tori.

Graph Classes	**Upper Bound**
A connected claw-free cubic	$\gamma_p(G) \leq n/4$
Block	$\gamma_p(G) \leq n/3$, with order $n \geq 3$
Petersen	$\gamma_p(P(n,k)) \leq min\{\lceil n/3 \rceil, k\}$, for $n \geq 4$
Cylinder	$\gamma_p(P_n \Box C_m) \leq min\{\lceil \frac{m+1}{4} \rceil, \lceil \frac{n+1}{2} \rceil\}$
Tori	$\gamma_p(G) \leq \begin{cases} \lceil \frac{n}{2} \rceil & \text{if } n \equiv 2 \bmod 4 \\ \lceil \frac{n+1}{2} \rceil & \text{otherwise} \end{cases}$

Table 3.2: The Approximate upper bounds for a Power Domination Set

Furthermore, Brueni and Heath [28, 29], and Zhao *et al*. [109] independently showed that there exists a power dominating set of size at most (n/3) for any graph with at least three vertices, and characterized the extremal graphs that attain the upper

bound. The *Power Domination* problem in Mycielskian and generalised Mycielskian graphs was investigated by Varghese, and Vijayakumar [103], they obtained closed formulae for the *Power Domination* number for Mycielskian of the complete graph, the wheel, the n-fan and n-star. Moreover, closed formulae for the *Power Domination* number are obtained for the Cartesian product of paths and cycles [20], and for direct product and strong product of path graphs are obtained in [37]. To the best of our knowledge, no further results are known for solving the PDS problem, either optimally or approximately.

3.3 PDS Algorithms

The basic problem of a PDS is a graph theory problem, which is related to the vertex covering and domination problems Haynes *et al*. [52]. Thus, the PDS problem is not related only to the power system industry, but also as a new problem in graph theory as Haynes *et al*. pointed out.

A set $S \subseteq V$ of a given graph $G = (V, E)$ is a *Power Dominating Set*, if every vertex and edge in G are observed by S, by applying the observation rules of power system monitoring. The minimum cardinality of a *Power Dominating Set* of G is denoted by $\gamma_p(G)$. The following reviews some PDS algorithms on different graph classes.

3.3.1 A PDS of Trees

Haynes *et al*. [52] provided a linear-time algorithm to solve a PDS in trees, where proved that if there is a tree T that has k vertices of degree of at least 3, then $\gamma_p(G) \geq (k+2)/3$. Furthermore, Haynes *et al*. showed that the following lemma is true:

Lemma 3.1 (*k* Vertices of Degree of at Least 3 in a PDS, Haynes *et al*. [52])
Every graph G having maximum degree of at least 3, there exists a $\gamma_p(G)$-set in which every vertex has a degree of at least 3.

Guo *et al*. [49] improved the results shown by Haynes *et al*. via developing a simpler linear-time algorithm for a PDS in trees. The theoretical properties of the *Power*

Dominating number in trees have been studied [52]. Their algorithm depends on finding a spider tree, whereby a tree with one vertex of degree of at least 3 and all others with a degree of at most 2.

Theorem 3.2 (A PDS of Trees, Haynes *et al.* [52])
For any tree T, $\gamma_p(T) = 1$ if and only if T is a spider.

Based on Theorem 3.2, another observation by Haynes *et al*. is that the number of each set of the partition T into a spider is defined as the number of a PDS in T. Hence:

Lemma 3.3 (The Spider Number of a Tree (T) by Haynes *et al.* [52])
The spider number of a tree T, denoted by $sp(T)$, to be the minimum number of subsets into which $V(T)$ can be partitioned so that each subset induces a spider such that for any tree T, $sp(T) \leq \gamma_p(T)$.

Lemma 3.4 (The Relationship Between a Spider and a PDS by Haynes *et al.* [52])
For any tree T, $\gamma_p(T) \leq sp(T)$.

Together with Lemma 3.3 and 3.4, Haynes *et al*. obtained Lemma 3.5, with an emphasis on the *Power Dominating* number of a tree is precisely equal to the spider number of the tree:

Lemma 3.5 (The Number of a PDS in Trees by Haynes *et al.* [52])
The Power Dominating *number of a tree is precisely the spider number of the tree such that for any tree T, $\gamma_p(T) = sp(T)$.*

Guo *et al*. enhanced the algorithm of Haynes and offered a much simpler linear-time algorithm, where their algorithm is based on a bottom-up strategy. All the inner nodes of T are arranged in the list L according to a post-order traversal of T; and then if v has at least two unobserved children then $P \leftarrow P \cup v$. After that, exhaustively apply the second observation rules to T. Finally, if r is unobserved then $P \leftarrow P \cup r$.

3.3.2 A PDS on Undirected Graphs with Tree-Width

Haynes *et al*. solved a PDS on trees on graphs with *tree-width* of one, the generalisation for a PDS in graphs of bounded *tree-width* has been left as an open question by Haynes *et al*. Therefore, Guo *et al*. [48] proposed an efficient algorithm to solve a PDS in polynomial time for input graphs that are "nearly" trees with a constant number of additional edges k, where a tree with k edges added has *tree-width* bounded by $k + 1$. Note that the class of graphs does not include all graphs with *tree-width* of two. As a graph of bounded *tree-width* can be formulated in monadic second-order logic and solvable in linear time as shown by Courcelle *et al*. [32], Kneis *et al*. [63] obtained the general result that a PDS is solvable in linear time for any fixed *tree-width* k, by formulating the PDS problem in the monadic second-order logic.

Moreover, a PDS is Fixed-Parameter Tractable (*FTP*) with respect to the parameter *tree-width* as shown by Kneis *et al*. [63], whereby the complexity class FTP denotes the solution of the problem in in $f(k)n^c$ steps, where c is a constant and f is an arbitrary function. As a result, Guo *et al*. [49] showed that the (undirected) PDS problem can be solved in $f(k).n$ time when k denotes the *tree-width* of the underlying graph, and a tree decomposition is given. They designed a dynamic programming algorithm to solve the PDS problem optimally in linear time on graphs of bounded *tree-width* via applying a new formulation called *valid orientation* of the edges. The algorithm is based on the following theorem:

Theorem 3.6 (A PDS on Undirected Graphs with Tree-Width, Guo *et al*. [49])
For a n-vertex graph with given width-k tree decomposition, a PDS can be solved in $O(nc^{k^2})$ *time for a constant c.*

Based on the formulation by Guo *et al*., Aazami [1] also provided a dynamic programming algorithm depending on a tree decomposition for solving the ℓ-round (undirected) PDS problem optimally in polynomial time on graphs of bounded *tree-width*, where the problem has the same observation rules of the PDS problem, except **OR2**, which applies the domination rule in *parallel* in at most $\ell - 1$ rounds.

3.3.3 A PDS on Directed Graphs

The PDS problem of directed graphs has been studied by Aazami and Stilp [2], where the DIRECTED PDS problem can be defined:

Definition 3.1 (A PDS on Directed Graphs, Aazami and Stilp [2])

Let G be a directed graph. Given a set of vertices $S \subseteq V(G)$, the set of vertices that are power dominated by S, denoted by $P(S)$, is obtained as follows:

D1: *If a vertex v is in S, then v and all of its* out-neighbours *are in $P(S)$.*

D2: *If a vertex v is in $P(S)$, one of its* out-neighbours *w is not in $P(S)$, and all other* out-neighbours *of v are in $P(S)$, then w is inserted into $P(S)$. This condition is called a Propagation rule.*

The authors of [2] developed a linear time algorithm to solve the DIRECTED PDS problem if the underlying undirected graph has bounded *tree-width*. The algorithm is based on the extension of the formulation of a PDS on undirected graphs introduced by Guo *et al*. [49]. Aazami and Stilp reformulated the notion of valid orientations to obtain the reformulation of a *Directed PDS* in terms of *Valid Colourings* of edges (similar to the formulation by [49], where blue and red edges play the same role as unoriented and oriented edges, respectively). The dynamic programming algorithm for a *Directed PDS* is based on the following lemma:

Lemma 3.7 (A PDS on Directed Graphs, Aazami and Stilp [2])

Given a directed graph G and $S \subseteq V(G)$, S power dominates G if and only if there is a Valid Colouring *of G with S as the set of origins, such that the running time of the algorithm is $O(nc^{k^2})$ where n denotes a set of tree nodes and c is a constant.*

3.3.4 A PDS in Block Graphs

A PDS in block graphs has been studied in [16, 107]. In [16], Atkins *et al*. achieved a bound of a PDS on block graphs by dividing a block graph into block spiders, relying on earlier results of Haynes *et al*. [52] in respect of finding the number of spiders. While in [107], Xu *et al*. applied a different way for solving the PDS problem through presenting a linear colour marking-based algorithm, which works on a

tree-like decomposition structure of a block graph. The following introduces basic definitions that we relied on while proposing the algorithms in this thesis:

Definition 3.2 (Cut-Vertex or Articulation Point)

Given an undirected graph G, a vertex $v \in V(G)$ is called a Cut-Vertex *or* articulation point *of G if by removal of v (and all its edges) G becomes disconnected.*

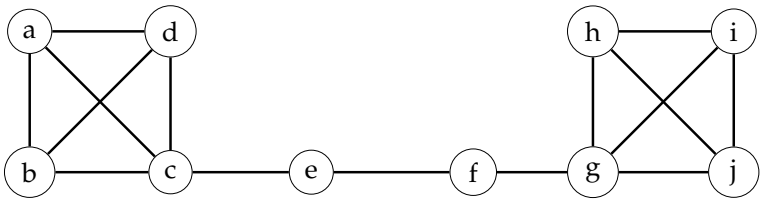


Figure 3.1: A Block of Graph G, taken from [16], where the two K_4 blocks $\{a, b, c, d\}$ and $\{g, h, i, j\}$ are End-Blocks because both have only one *Cut-Vertex*, namely $\{c, g\}$

Definition 3.3 (A Block of a Graph, [51])

A block of a graph is a maximal non-separable subgraph. An example is given in Figure 3.1.

Definition 3.4 (An End-Block)

An End-Block of G is a block that contains only one Cut-Vertex *of G.*

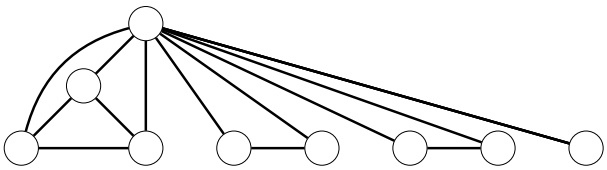


Figure 3.2: An Example of A Block-Star, taken from [16], where white circles imply vertices

Definition 3.5 (Block-Star)

Let G be a block graph. A graph G is called Block-Star if G itself is a block or if every block of G is an End-Block. An example of a Block-Star graph is given in Figure 3.2.

Notice that a Star $K_{1,n}$ where $n \geq 1$ is simply a Block-Star where every block is a K_2 block.

Definition 3.6 (A Spider Graph)

The graph of a Spider is a tree T that has at most one vertex of degree 3 or more and all others with degree at most 2. An example of the graph of a Spider is given in Figure 3.3.

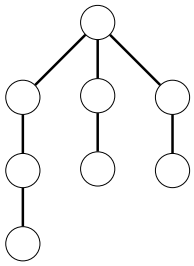


Figure 3.3: An Example of a Spider Graph, taken from [16], where white circles denote vertices

Definition 3.7 (A Block-Spider)

A Block-Spider is defined as a Block-Star, if assigned a path to all or to some of its vertices so that the resulting paths are vertex-disjoint (see Figure 3.4).

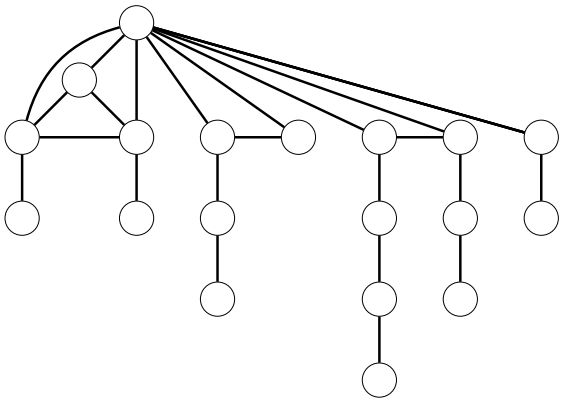


Figure 3.4: An Example of a Block-Spider, taken from [16]

Therefore, a maximal connected induced subgraph without a *Cut-Vertex* is called a block of G (i.e. every block in G is complete), where for each a block graph of G, if there exists only one *Cut-Vertex*, then it is called an End-Block of G. The relationship between a PDS and a block graph is studied in [16] where the main result is based on the following lemma:

Lemma 3.8 (A PDS in Block Graphs, Atkins *et al.* [16])

For any block graph G, $\gamma_p(G) = 1$ if and only if G is a Block-Spider.

However, a linear time algorithm of finding a minimum PDS in block (undirected) graphs has been designed by Xu *et al*. [107]. The authors of [107] proved that for each *Cut-Vertex* of G there exists a PDS.

Lemma 3.9 (A Cut-Vertex in Block (Undirected) Graphs, Xu *et al.* [107])

Let G be a block graph, there exists a $\gamma_p(G)$-set in which every vertex is a Cut-Vertex *of G.*

The algorithm is based on a tree-like decomposition structure of a block graph, using the *Depth First Search* method to build a cut-tree (an ordinary tree) of a block graph in linear time. Note that the input of the problem is a cut-tree of a block graph, and during the algorithm a Block-vertex is processed as a vertex, where each Block-vertex contains a subset of vertices of the block graph.

3.4 Network Controllability under Vulnerability

The ability of an attacker to take over control of a distributed system or to deny the defender is a general problem, but of particular significance in cyber-physical systems where even temporary *loss of view* or *loss of control* can result in outright failure and severe cascading effects. Moreover, many such cyber-physical systems not only exhibit a safe *fail-stop* behaviour such that they can be brought to a halt in a safe state, but also have hard real-time requirements such as in the case of electrical power networks and their constituent elements. This offers a strong motivation to study the ability of such systems to recover from deliberate attacks, leading to the decrease of network performance due to a selected removal of vertices or edges.

Networks comprising nodes and edges represent the foundation of a variety of social, environmental and technological systems. In complex networks, individuals constitute the nodes, whereas the relationships between them constitute the edges. The attack vulnerability of a range of complex network models such as the Erdős-Rényi random graph, real world networks and in which a portion of the nodes or edges was eliminated, was measured by a number of recent studies on random failures and malicious attacks on complex networks [91, 6, 22, 104, 53, 45, 77]. The effect of network *controllability* of directed Erdős-Rényi and scale-free networks under attacks and cascading failures has been studied by Pu *et al*. [91]. The authors of [81] reviewed two issues regarding the security of complex networks. The first issue is related to the removal of vertices due to random or intentional attacks, which

can trigger a series of overload failures, leading to the partial or complete collapse of the network. The second issue pertains to range-based attacks on edges, which is especially relevant as the majority of complex network security studies have focused on attacks on nodes and not on edges.

Consequently, protection is the ultimate goal of studies on attack vulnerability. In order to ensure network protection by shielding or a temporary isolation of some vertices or edges, it is important to determine the key vertices or edges, the elimination of which leads to the entire network malfunctioning. In addition, it is necessary to learn how to create networks able to withstand attacks as well as to maintain the capability to control the systems. The following reviews the most attack vulnerabilities that have been extensively studied on some complex networks with emphasis on (directed/undirected) Erdős-Rényi random graph in order to show how the absence of key vertices and edges can impact on a control graph.

3.4.1 The Vulnerability under Vertex and Edge Attacks

Caused by random failures or malicious attacks, the malfunctioning of some network vertices can result in the breakdown of the entire network into isolated parts. As we focus on Erdős-Rényi random graph, in particular directed graphs, the most important study was proposed by Pu *et al*. [91], where they investigated the *controllability* of directed Erdős-Rényi and scale-free networks under *Single-node attack* and *Cascading failure attack*. The authors of [91] found that the efficiency of degree-based attacks on network *controllability* is greater than that of random attacks on network *controllability* for directed Erdős-Rényi and scale-free networks; at the same time, network *controllability* is also adversely affected by cascade failures, even when induced by a local node failure. The implications of eliminating vertices from various networks when exposed to a range of types of attack were investigated by the authors of [53], where they studied the attack vulnerability of six different complex networks including Erdős-Rényi model of random networks. They also observed that compared to original network-based attack strategies, elimination by the recalculated degrees and betweenness centralities was more deleterious.

However, Barthlemy [22] analysed the significance of the betweenness centrality of nodes in Erdős-Rényi and scale-free networks where the removal of betweenness centrality of nodes results in the emergence of new disconnected components. A novel method of improving network robustness against high-degree vertex removal was proposed by Schneider *et al*. [95] who studied both Erdős-Rényi and scale-free networks. The control centrality of single node in complex networks such as directed Erdős-Rényi random graph was investigated by Liu *et al*. [72] where the presented results proved that "randomly selected upstream (or downstream) neighbours have more outgoing (or incoming) links than the node itself". This, however, allowed to understanding the *controllability* of complex networks and design an efficient attack strategy against network control.

In addition to vertices, it is also possible to attack edges. However, in computer networks attacks on vertices may take the form of breakdowns of servers by malicious attackers, attacks on edges involve disconnecting communication cables [53]. The other significant study based on the *controllability* of directed Erdős-Rényi was carried out by Nie *et al*. [85]. The authors [85] investigated the robustness of the *controllability* of directed Erdős-Rényi networks based on various points of the cascading failures, as a result of the elimination of the highest load edge, and showed that *controllability* for networks evolves during cascading failures in the case of two distinct attack strategies, random and intentional.

A study on how susceptible different complex networks such as the Erdős-Rényi model of random networks are to a range of edge attacks was undertaken by the author of [53]. Another study by [75] explored the addition of edge directions, depending on the node residual degree, as a method of improving complex network *controllability*; based on this, a technique of designating edge direction was suggested to show the effectiveness of the proposed method on the two basic network models Erdős-Rényi and scale-free networks. On the other hand, Wang [104] conducted an investigation of how *structural controllability* can be maintained during malicious attacks on directed networks, merging the issue of the control robust-

ness into the problem of transitivity maximisation for control routes and proposing an efficient greedy algorithm to create transitive control routes.

3.5 Analysis of Embedding Structures in Directed Erdős-Rényi Graphs

Most existing works in this chapter have been considered (undirected) graphs. Our objective is to identify a possible embedding of such structures in directed Erdős-Rényi graphs of different density (i.e. allowing for some removal of edges or nodes owing to attacks or failures) in order to be adapted to solve the DIRECTED PDS problem if a control network has been partitioned or damaged.

Guo *et al*. [49] developed a linear-time dynamic programming algorithm based on valid orientations for optimally solving PDS on graphs of bounded *tree-width*, introducing the notion of valid orientations for a new formulation of PDS (over undirected graphs). Based on this orientation by Guo *et al*., Aazami and Stilp [2] reformulated *Directed PDS* in terms of *Valid Colouring* of the edges in order to develop an algorithm based on dynamic programming for a *Directed PDS*. The aim of the *Valid Colouring* is to model the application of rules (**D1**) and (**D2**) of a *Directed PDS* as defined in Definition 3.1. We therefore propose a reconstruction algorithm based on recent work by Guo *et al*., Aazami and Stilp as well as for directed control graphs in Erdős-Rényi graphs arising after attacks (more details in Chapter 4) . This algorithm assumes that the underlying undirected graph has bounded *tree-width* constructed as a nice tree decomposition yielding a best-case complexity of $\mathcal{O}(nc^k)$, a worst-case complexity of $\mathcal{O}(nc^{k^2})$, and an average complexity of $\mathcal{O}(\log nc^{k^2})$ [13].

The linear time algorithm for solving a PDS in Block graphs is developed by Xu *et al*. [107] where it is based on a cut-tree of a block in undirected graphs constructed in linear time by *Depth-First Search* (DFS). Therefore, we could take the advantage of *DFS* for constructing a tree-like structure on the directed Erdős-Rényi graph, which gives equivalent characterisations of trees by considering tree edges, and classifies the edges of a digraph into four edge types (i.e. a tree edge, a back

edge, a forward edge and cross edge); this, however, requires a relatively high density. Subsequently we propose a novel algorithm [14] based on re-using as much as possible of remaining fragments from the original control graph where permitted whilst identifying unutilised edges to minimise the number of a PDS, offering *controllability* after an event or attack leading to the partitioning of the original control network (further discussion in Chapter 5). This DFS-based approach yields an improved average-case complexity over a reconstruction algorithm for (directed) control graphs proposed in [13].

Furthermore, the linear time algorithm propose by [16, 107] for solving a PDS in block (undirected) graphs is based on identifying a possible *Cut-Vertex* (or an *articulation point*) where a *Cut-Vertex* or an *articulation point* is a vertex whose removal (together with the removal of any incident edges) results in a disconnected graph. Although the study of a PDS of block (directed) graphs introduces differences such as directed edges that should be considered separately. Therefore, it is helpful to build a tree-like decomposition structure of a block (directed) graph assuming that given a weakly connected graph, such that the undirected underlying graph can form an ordinary tree. This is regardless of the fact that every Block-Vertex is actually a subset of vertices of the original block graph. We therefore propose a novel algorithm to re-construct a control graph as far as possible in the presence of *compromised nodes*. The approach is based on a BLOCK DECOMPOSITION of a directed graph, allowing us to re-construct a PDS structure by applying three phases (further details in Chapter 6).

3.6 Summary

This chapter reviewed the problems of *controllability* and *structural controllability* as represented by the PDS problem and reviewed algorithms for specific classes of graphs for which improved efficiency relative to the general case is known. Our interest lies primarily in finding an efficient embedding for Erdős-Rényi graphs of different density, and to adapt approaches discussed here for the case of a PDS over directed Erdős-Rényi graphs.

Chapter 4

A New Algorithm for a Power Dominating Set

4.1 Overview

We reviewed the problems of *controllability* and *structural controllability* as represented by the PDS problem in Chapter 2; followed by a summary of existing algorithms for specific classes of graphs, for which a PDS has been studied before, to identify a potential embedding of such structures in Erdős-Rényi graphs in Chapter 3.

In this chapter we propose a reconstruction algorithm for (directed) control graphs of bounded *tree-width* embedded in Erdős-Rényi random graphs particularly in partitionings of the original graphs based on recent work by Aazami and Stilp as well as Guo *et al*. We therefore design a new algorithm to compute a PDS for a given directed graph when the controllability is compromised such as an adversary with sufficient knowledge of the network distribution and its power domination can disconnect parts of the control original graph and leave parts of a system uncontrolled. This dynamic programming algorithm is based on a *nice tree decomposition* for a given graph, and therefore, we propose the *7-Colourings* of a *dependency path* reflecting the direction of edges and their colours in order to concatenate *dependency paths* and detect *dependency cycles* while computing bags in a *nice tree decomposition*. This also yields to reduce the number of combinations required to compute a PDS at each bag where there are at most $(7^{(k+1)^2} \cdot 5^{k+1} \cdot 2^{(k+1)^2})$ states for each bag X_i with $|X_i| \leq (k+1)$.

4.2 Problem Statement and Assumption

Given a directed graph $G^{'} = (V, E)$, constructed as $ER(n, p)$, where the underlying undirected graph has bounded *tree-width*. We assume that $G^{'}$ satisfies the assumptions (1,2 and 3) as shown in Subsection 1.4.2 in Chapter 1. A real-world scenario related to this problem is precisely the industrial automation control systems (e.g. SCADA systems) which deploy their elements following a mesh distribution to monitor other critical infrastructures (e.g. power systems), where $G^{'} = (V, E)$ depicts the network distribution with V illustrating the elements (e.g. remote terminal units, servers, etc.), and E representing the communication lines. In this context, the interpretation of a PDS would be $G^{'} = (V, E)$ which represents a power network containing a set of electrical or control vertices (a substation bus where transmission lines, loads, and generators are connected) and a set of edges connecting a transmission line or a communication link joining the two electrical vertices.

Let $S^{'}$ be a given PDS of $G^{'}$ and assume that attackers in a position to eliminate some vertices of $G^{'}$ (i.e. in real-world context when an electric actuator or industrial sensor in power systems are subjected to intentional or random removal such as attackers can control a subset of sensors or actuators that have the ability to control more states, and therefore, act as Man-in-the-Middle between remote terminal units and its elements in an electrical power network). This deletion of vertices may result in a disconnected component of a directed graph $G^{'}$, defined by $H = (V, E)$, where $H = (V, E) \subset G^{'}$ is partitioned from the original graph $G^{'}$ such that $V(H) \notin V(G^{'})$ and $E(H) \notin E(G^{'})$ (i.e. there exists no edge in H whose one end vertex is in $G^{'}$ and verse vice). In this case, a PDS of a directed graph $H = (V, E)$ (i.e. a disconnected component of $G^{'}$) may be different to the remainder of $S^{'}$.

Throughout this chapter, we design a dynamic programming algorithm, satisfying the assumption (4) in Subsection 1.4.2 in Chapter 1, to reconstruct a PDS for a given directed graph $H = (V, E)$ when the underlying undirected graph has bounded *tree-width*. Note that $H = (V, E)$ is a disconnected component of $G^{'}$ as a result of the scenario mentioned above.

•Input:

As proved by Kloks [62], a *tree decomposition* of width k can be transformed to a *nice tree decomposition* with k in linear time. Thus, we assume that a *nice tree decomposition* of H is given where the underlying undirected graph has bounded *tree-width* and H satisfies the same assumptions (1,2 and 3) as $G^{'}$ mentioned in Subsection 1.4.2 in Chapter 1.

•Question:

Can we design an algorithm to reconstruct a PDS for H, denoted by S (*e.g.* in real-world monitoring system, can we reconstruct as few measurement devices that have the ability to control more states as possible in an electric power system when an attacker able to affect the physical state of the system by compromising a subset of sensors or actuators that can disconnect parts of the control graph).

•Output:

A minimal $S \subseteq V$ for H by applying a *Valid Colouring* of edges such that all vertices in $V \in H$ are power dominated by the vertices in S.

4.3 PDS Tree Decomposition

Guo *et al*. [49] developed a linear-time dynamic programming algorithm based on valid orientations for optimally solving a PDS on undirected graphs of bounded *tree-width*, introducing the notion of valid orientations for a new formulation of a PDS. Computational complexity is dominated by determination of the mapping (A_i) for a join node, where for each bag state (s) it is required to consider all pairs of compatible bag states of its two children. Therefore, the running time algorithm proposed by Guo *et al*. is $\mathcal{O}(nc^{k^2})$ where n denotes a set of tree nodes and c is a constant. Based on this orientation by Guo, Aazami and Stilp [2] reformulated a *Directed PDS* in terms of a *Valid Colouring* of edges to develop an algorithm based on dynamic programming for a *Directed PDS*. This algorithm is applied to directed graphs such that the underlying undirected graph has bounded *tree-width*. The aim of a *Valid Colouring* is to model the application of rules (**D1**) and (**D2**) of a *Directed PDS*. Both algorithms are based on tree decompositions of graphs and their use

with respect to dynamic programming. We now introduce some basic definitions:

Definition 4.1 (Tree Decomposition)

For $G = (V, E)$, tree decomposition *of G is a pair $\langle X_i | i \in I, T \rangle$, where each X_i is a subset of V, called a* bag, *and T is a tree with the elements of I as nodes satisfying:*

a. $\bigcup_{i \in I} X_i = V$.

b. For every edge $uv \in E$ has both ends in some X_i such that $\{u, v\} \subseteq X_i$.

c. $\forall i, j, k \in I$: *If j is on the unique path from i to k in T, then $X_i \cap X_k \subseteq X_j$.*

Definition 4.2 (Tree-Width)

The width *of a* tree decomposition $\langle X_i | i \in I, T \rangle$ *is defined as* $\max_{i \in I} |X_i| - 1$. *The* tree-width *of G, denoted by $tw(G)$, is defined as the minimum* width k *over all* tree decompositions *such that G has a* tree decomposition *of* width k. *The nodes of T are called T-nodes and X_i bags.*

We subsequently rely on the special case of *nice tree decompositions* to simplify the design of the dynamic programming algorithm.

Definition 4.3 (A Nice Tree Decomposition)

A nice tree decomposition $\langle X_i | i \in I, T \rangle$ *for a graph $G = (V, E)$, where T is a rooted tree, is a* tree decomposition *for G if the following conditions are satisfied:*

1. *Every node of the tree T has at most 2 children.*
2. *The nodes i of T are one of four node types:*
   a. ***Leaf Nodes** without children and corresponding leaf bags X_i have $|X_i| = 1$.*
   b. ***Forget Nodes** have one child j with $X_j = X_i \cup \{v\}$.*
   c. ***Introduce Nodes** with one child j where $X_i = X_j \cup \{v\}$.*
   d. ***Join Nodes** i have two children $j, k \in I$ with $X_i = X_j = X_k$.*

For a graph of width k, Lemma 4.1 due to Kloks [62] can be drawn as follows:

Lemma 4.1 (A Nice Tree Decomposition, Kloks [62])

Given a tree decomposition *of a graph $G = (V, E)$ of width k and $\mathcal{O}(n)$ nodes, where n is the number of nodes in G, one can find a* nice tree decomposition *of G that has $\mathcal{O}(n)$ nodes and the same width k in time $\mathcal{O}(n)$.*

Based on the valid orientation of undirected graphs proposed by Guo *et al*. [49], Aazami and Stilp [2] introduced the reformulation of a *Directed PDS* in terms of a *Valid Colouring* of edges. Consequently, we reformulate a PDS of a directed graph $H = (V, E)$ in terms of a *Valid Colouring*, which is similar to the formulation by [49], where blue and red edges play the same role as unoriented and oriented edges, respectively. Our approach applies to a directed graph such that the underlying undirected graph has bounded *tree-width*.

Definition 4.4 (The Colouring of a Directed Graph)

A colouring *of a directed graph $H = (V, E)$ is a partition of the edges in H into red and blue edges. We denote a* colouring *by $C = (V, E_r \cup E_b)$ where E_r is the set of red edges and E_b is the set of blue edges.*

Definition 4.5 (A *Valid Colouring*, Aazami and Stilp [2])

A Valid Coloring *$C = (V, E_r \cup E_b)$ of a directed graph $G = (V, E)$ is a* coloring *of G with the following properties:*

1. *No two antiparallel edges can be coloured red.*
2. *The subgraph induced by the red edges, $G_r = (V, E_r)$, has the following properties:*
   a) *$\forall v \in G : d^-_{G_r}(v) \leq 1$, and*
   b) *$\forall v \in G : d^-_{G_r}(v) = 1 \implies d^+_{G_r}(v) \leq 1$.*
3. *G has no* dependency cycle. *A* dependency cycle *is a sequence of directed edges whose underlying undirected graph forms a cycle such that all the red edges are in one direction, all the blue edges are in the other direction, and there are no two consecutive blue edges.*

We now define a *Valid Colouring* for a directed graph $H = (V, E)$ based on the reformulation of the DIRECTED PDS problem proposed by Aazami and Stilp; informally speaking, these colorings model the application of rules (**D1**) and (**D2**) of a *Directed PDS* (see Definition 2.16):

Definition 4.6 (A *Valid Colouring* for a Directed Graph)

A Valid Coloring *$C = (V, E_r \cup E_b)$ of a directed graph $H = (V, E)$ is a* coloring *of H with the following properties:*

1. *No two antiparallel edges can be coloured red.*
2. *The subgraph induced by the red edges, $H_r = (V, E_r)$, has the following properties:*
   a) $\forall v \in H : d^-_{H_r}(v) \leq 1$, *and*
   b) $\forall v \in H : d^-_{H_r}(v) = 1 \Longrightarrow d^+_{H_r}(v) \leq 1$.
3. *H has no* dependency cycle. *A* dependency cycle *is a sequence of directed edges whose underlying undirected graph forms a cycle such that all the red edges are in one direction, all the blue edges are in the other direction, and there are no two consecutive blue edges.*

We call a vertex with $d^-_{H_r}(v) = 0$ in $H_r = (V, E_r)$ is an origin of C, where H_r is denoted by the set of vertices with only red edges. Therefore, an origin of *Valid Colouring* of H is defined as:

Definition 4.7 (The Origins of a *Valid Colouring* for a Directed Graph)
An origin vertex v of a Valid Coloring $C = (V, E_r \cup E_b)$ *of a directed graph $H = (V, E)$ is said to be a vertex in a* Directed PDS *if:*

1. *It either has no in-degree in H, where $v \in H : d^-(v) = 0$, or*
2. *it has no red in-coming edges in H^r such that $v \in H^r : d^-(v) = 0$, where H^r denotes the set of vertices with only red edges.*

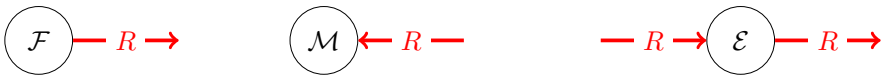
We define the connection between a *directed path* and a *dependency path* as follows:

Definition 4.8 (A Dependency Path in a *Valid Colouring*)
A dependency path, *denoted by P, from u to v is a path where all red edges in P are directed from u to v and all blue edges are directed from v to u. Therefore, a* dependency path *from u to v is a* directed path *with only directed red edges.*

We obtain a *dependency path* after applying a *Valid Colouring* to a directed graph; informally speaking, these colorings model the application of rules (**D1**) and (**D2**) of a *Directed PDS* (see Definition 2.16) which result in the formulation of the *structural controllability* for a graph. The dynamic programming algorithm for a *Directed PDS* is based on Lemma 4.2 due to Aazami and Stilp with valid orientations are replaced by viewing blue edges as unoriented edges, and red edges as oriented edges.

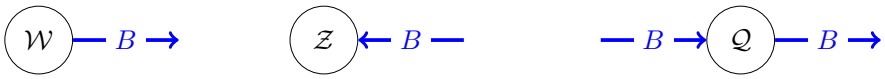
Lemma 4.2 (A *Valid Colouring* of a Directed Graph, Aazami and Stilp [2])
Given a directed graph G and $S \subseteq V(G)$, S power dominates G if and only if there is a Valid Colouring *for G with S as the set of* origins.

During executing a *Valid Colouring*, there is a possibility of existence a *dependency cycle*. Therefore, we seek to reduce the number of combinations required to compute a PDS at each bag through applying the *7-Colourings* of a *dependency path* based on the degree constraints of a *Valid Colouring* of a directed graph $H = (V, E)$. This *7-Colourings* approach yields to concatenate *dependency paths* and detect *dependency cycles* when computing bags in a *nice tree decomposition* by colouring the vertices in a *dependency path* as defined in the following definition:



(a) (b) (c)

Figure 4.1: The Colouring of Vertices with in/out Red Edges in a *Dependency Path*



(a) (b) (c)

Figure 4.2: The Colouring of Vertices with in/out Blue Edges in a *Dependency Path*

(a) (b)

Figure 4.3: The Colouring of Vertices with in/out Blue/Red Edges. Note that two different directions of (in/out) blue/red edges have the same colour

Definition 4.9 (The *7-Colourings* of a Dependency Path)
To detect dependency cycles, *we define seven colours for every vertex $v \in H^{r/b}$ in a* dependency path *depending on the directions of in/out blue and red edges where $H^{r/b}$ denotes the set of vertices with red or edges respectively. A* colouring *of vertices in a* dependency path *of a directed graph $H^{r/b} = (V, E_r \cup E_b)$ is assigning one of the colours $\{\mathcal{F}, \mathcal{M}, \mathcal{E}\}$ to each vertex with in/out red edges, and $\{\mathcal{W}, \mathcal{Z}, \mathcal{Q}\}$ to each vertex with in/out blue edges and $\{\mathcal{L}\}$ to each vertex with in/out blue and red edges (or vice versa) such that a vertex v is assigned to: (see Figures 4.1, 4.2 and 4.3, respectively.)*

1. *$\mathcal{F}$ if there exists a vertex with **no** in-coming edge and at least one out-going red edge:*
$$\exists v \in H^{r/b} : \Big((d^-_H(v) = 0\Big) \wedge \Big(d^+_{H^r}(v) = 1)\Big) \Longrightarrow \mathcal{F}.$$
2. *$\mathcal{M}$ if there exists a vertex with one in-coming red edge and **no** out-going edge:*
$$\exists v \in H^{r/b} : \Big((d^-_{H^r}(v) = 1\Big) \wedge \Big(d^+_H(v) = 0)\Big) \Longrightarrow \mathcal{M}.$$
3. *$\mathcal{E}$ if there exists a vertex with one in-coming red edge and one out-going red edge:*
$$\exists v \in H^{r/b} : \Big((d^-_{H^r}(v) = 1\Big) \wedge \Big(d^-_{H^r}(v) = 1)\Big) \Longrightarrow \mathcal{E}.$$
4. *$\mathcal{W}$ if there exists a vertex with **no** in-coming edge and one out-going blue edge:*
$$\exists v \in H^{r/b} : \Big((d^-_H(v) = 0\Big) \wedge \Big(d^+_{H^b}(v) = 1)\Big) \Longrightarrow \mathcal{W}.$$
5. *$\mathcal{Z}$ if there exists a vertex with one in-coming blue edge and **no** out-going edge:*
$$\exists v \in H^{r/b} : \Big((d^-_{H^b}(v) = 1\Big) \wedge \Big(d^+_H(v) = 0)\Big) \Longrightarrow \mathcal{Z}.$$
6. *$\mathcal{Q}$ if there exists a vertex with one in-coming blue edge and one out-going blue edge:*
$$\exists v \in H^{r/b} : \Big((d^-_{H^b}(v) = 1\Big) \wedge \Big(d^-_{H^b}(v) = 1)\Big) \Longrightarrow \mathcal{Q}, \text{ and}$$
7. *$\mathcal{L}$ if there exists a vertex with one in-coming red edge and one out-going blue edge or with one in-coming blue edge and one out-going red edge:*

$$\exists v \in H^{r/b} : \Big((d^-_{H^r}(v) = 1) \wedge (d^+_{H^b}(v) = 1)\Big) \vee \Big((d^-_{H^b}(v) = 1) \wedge (d^+_{H^r}(v) = 1)\Big) \Longrightarrow \mathcal{L}.$$

Throughout this thesis, if there exists a blue/red edge between v and u, then both vertices are connected. Therefore, we say that v is controlled (i.e. covered or power dominated) by u if there is a red edge from u to v; however, if there exists a blue edge $u \to v$, then v is not controlled by u.

Together with Definition 4.9, we immediately define a *dependency path* in a *Valid Colouring* of a directed graph $H = (V, E)$ as:

Definition 4.10 (A Dependency Path in a *Valid Colouring* of a Directed Graph) *A* dependency path *(P) in a* Valid Colouring *of H is a sequence of vertices with colours $\mathcal{F}$, $\mathcal{M}$ and $\mathcal{E}$ (i.e. a sequence of red edges) such that $P = v_1, e_1, v_2, e_2, \ldots, e_{i-1}, v$ has no vertex with the colour $\mathcal{Q}$ (i.e. no two consecutive blue edges) and P starts with a vertex (v_1) with the colour $\mathcal{F}$ and ends with a vertex (v_i) with the colour $\mathcal{M}$. Thus, all in-between vertices $v_2, \ldots, v_{i-1}$ in P are coloured with $\mathcal{F}$. A* dependency cycle *in a directed graph is a sequence of directed edges whose underlying undirected graph forms a cycle whose vertices have the colours $\mathcal{F}$, $\mathcal{E}$ and $\mathcal{M}$ (i.e. all red edges in one direction), and two vertices with the*

colour $\mathcal{L}$ (i.e. all blue edges in the other direction). Consequently, there is no vertex with colour $\mathcal{Q}$ (i.e. no two consecutive blue edges) (see Figure 4.4).

The following theorem shows that the number of colours required to check a dependency cycle while applying a *Valid Colouring* is exactly seven colours:

Theorem 4.3 (The 7-*Colourings* of a Dependency Path in a *Valid Colouring*) *Given a* dependency path P*, one can decide if P is a path or has a cycle, by colouring the vertices in the path with only seven colours.*

Proof. Supposing that there are only six colours constructed as follows: there exists a *dependency path* P, where P is a sequence of red edges $P = v_1, e_1, v_2, e_2, \ldots, e_{i-1}, v_i$ such that the first vertex v_1 of P has exactly one out-going red edge and the last vertex v_i has exactly one in-coming red edge and the in-between vertices have in/out red edges. Note that the directions of red edges for each vertex in P are categorised as three directions such that:

1. A vertex with an out-going red edge.
2. An in-coming/out-going red edge.
3. An in-coming red edge.

Thus, in order to distinguish between these three vertices, each vertex should be coloured differently according to the direction of the red edge (see Figure 4.1).
Now we apply the same argument above taking into consideration that the path given starts from an out-going blue edge, and ends with in-coming blue edges and the in-between vertices have in/out red edges such that:

1. A vertex with an out-going blue edge.
2. An in-coming/out-going red edge.
3. An in-coming blue edge.

Consequently, there exists only two different directions of blue edges such that the first vertex v_1 of P has exactly one out-going blue edge (see Subfig 4.2.(a)), and the last vertex v_i has exactly one in-coming blue edge (see Subfig 4.2.(b)). Therefore, both vertices can be assigned with a different colour. Hence, the sum of

colouring vertices is five so far if we exclude the duplication of an in-coming/out-going red edge for each case. Now we prove by way of contradiction that five colours is not sufficient to define *dependency paths* and detect *dependency cycles*. Given two paths P_{ux} and P_{xv} in order to concatenate them. Supposing that one could colour vertices in each path with at most five colours such that the colour of the first vertex has to be one of $\{\mathcal{F}, \mathcal{W}\}$ and the last vertex should be one of $\{\mathcal{Z}, \mathcal{M}\}$ based on the directions of in/out blue and red edges that are incident from/to vertices, and in-between vertices should be coloured with $\{\mathcal{E}\}$. Note that $v \in p_{ux} \vee p_{xv} \subseteq \{\mathcal{F}, \mathcal{M}, \mathcal{E}, \mathcal{W}, \mathcal{Z}\}$. After concatenating the two given paths, the vertex x has three cases of in-coming and out-going edges:

i. $d^-(x) = blue \wedge d^+(x) = red$,

ii. $d^-(x) = red \wedge d^+(x) = blue$, or

iii. $d^-(x) = blue \wedge d^+(x) = blue$.

Assuming that the obtained path is not a *dependency path* meaning the vertex x has in/out blue edges. This is a contradiction, in reality, there is no colour in the set above can reflect the direction of blue edges. Thus it could assign a colour $\mathcal{Q}$ to the vertex (see Subfig 4.2.(c)).

On the other hand, supposing the obtained path is a *dependency path* meaning the vertex x has in-blue and out-red edges or vice versa. This is also a contradiction, since there is also no colour in the set above that can reflect the direction of blue/red edges. Thus it could assign a colour $\mathcal{L}$ to the vertex (see Figure 4.3). Hence, by combining all the colours for each vertex of P depending on each different direction of blue and red edges, then there are exactly seven colours that can determine whether P is a *dependency path* or has a *dependency cycle*. Thus seven colours are required to sufficiently detect a *dependency cycle* to concatenate paths.

However, we define only one colour for in-blue/out-red edges or vice versa (see Figure 4.3). Assuming that the edges of $\mathcal{C}$ are alternatively blue and red (starting with blue edge or vice versa). As a vertex has in-blue and out-red or (vice versa), the colour for both is the same. The reason is that a path is a cycle even if it has alternatively blue and red or vice versa until it has a vertex with a colour $\mathcal{Q}$ (i.e. there

are no two consecutive blue edges) (see Figure 4.4). Therefore, it is not important to have an independent colour for the different directions of in-blue and out-red edges or vice versa.

We extend the result of Aazami and Stilp, which leads to the formulation of a dynamic programming algorithm for a directed graph $H = (V, E)$:

Theorem 4.4 (A Valid Colouring of a Directed Graph)
Given a directed graph $H = (V, E)$ and $S \subseteq V(H)$, S power dominates H if and only if there is a Valid Colouring *of H with S as the set of* origins.

Proof. Supposing that $S \in V(H)$ is a solution to a PDS of a directed graph $H = (V, E)$, satisfying the same assumptions (1,2 and 3) as $G^{'}$ shown in Subsection 1.4.2 in Chapter 1, such that the underlying undirected graph has bounded *tree-width*; thus, $P(S) = V(H)$. We apply a *Valid Colouring* C with S as the set of *origins* by colouring the edges in H according to the degree constraints in Definition 4.6. We colour an edge (uv) red from u toward v if:

1. v is covered by applying the domination rule (**D1**) to u where $u \in S$, or
2. v is covered by applying the propagation rule (**D2**) to u.

Note the domination rules **D1** and **D2** should be in order (i.e. we first apply the rule **D1** on u to cover all neighbours of u that are not power dominated yet, and only after that we apply the propagation rules **D2**). Moreover, we do not apply **D1**, **D2** to cover previously covered vertices. Thus, it can be verified with this *colouring* the degree constraints mentioned in Definition 4.6 are satisfied, where each vertex in $V_D \in S$ can control at least two out-going red edges, and each vertex in V_S can control at most one out-going red edge and the propagation rule can be applied to a vertex $v \in V \setminus S$ and power dominates at most one of the neighbours of v.

We show by way of contradiction that there is no *dependency cycle* in a *Valid Colouring*. Let $u \to v$ denotes a vertex v is covered after a vertex u; assuming that $C = u_1, u_2, \ldots, u_m$ is a *dependency cycle* and all red edges in C are in the same direction, then the red edges (u_i, u_{i+1}) imply that $u_i \to u_{i+1}$ for all $i = 1, 2, \ldots, m-1$; thus we get $u_1 \to u_2 \to \cdots \to u_m$, but this is a contradiction since the last red

edge from u_m back to u_1 implies $u_m \to u_1$. Hence, there is no *dependency cycle* with all edges coloured red.

Supposing that the *dependency cycle* C has some blue edges (see Figure 4.4). There exists a contradiction when there are no two consecutive blue edges. Assuming that the *dependency cycle* C is even, and its edges are alternatively blue and red (starting with a blue edge); when combining these dependencies, we find that $u_1 \to u_3 \to u_1$ then we see that $u_1 \to u_1$ and this gives the contradiction.

Now we prove that a directed graph $H = (V, E)$ has a *Valid Colouring* $C = (V, E_r \cup E_b)$ with $S \subseteq V(H)$ as the set of *origins*. Note that the vertices in S and all vertices $u \in N(S)$ (i.e. their *out-neighbours*) in $H^r = (V, E_r)$ are covered by applying the rule (D1). We show that S will power dominate all vertices in H. Assuming that S does not cover all vertices in H (i.e. $P(S) \neq V(H)$). Let $X \subset V$ be the maximal set of vertices that can be covered by S; that is, $P(S) = X$. We claim that there is at least one red edge from X to $V \setminus X$. Note that all of the *origins* are in X, so each of the vertices in $V \setminus X$ has an in-degree of 1 in $H^r = (V, E_r)$. Thus, if there is no red edge from X to $V \setminus X$, then there should be a directed cycle of red edges in $H[V \setminus X]$. This is not possible since there are no *dependency cycles*. Let $e1 = (u_1, v_1), \ldots, e_k = (u_k, v_k)$ be all of the red edges from X to $V \setminus X$. Supposing that there exists $v_i \notin X$, then v_i can be covered by applying rule $(D2)$ to u_i. Owing to the maximality assumption of X this is not allowed. Therefore, each u_i has another *out-neighbour*, such as z_i, in $V \setminus X$. If (u_i, z_i) is a blue edge, then:

i. u_i has an in-degree of 1 and z_i is covered by applying $(D2)$ to u_i, or

ii. u_i is a vertex with an out-degree of at most 1 (i.e. an *origin*) and z_i would be controlled by applying rule $(D1)$ to u_i.

Therefore, we have $X = V$, so S power dominates H.

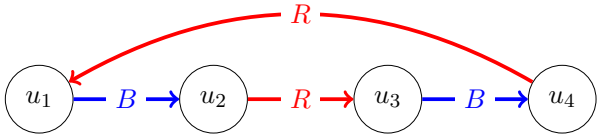


Figure 4.4: A *Dependency Cycle*

4.4 Colouring and Repair Algorithms

We describe the formal definition of the algorithm and proof of correctness for solving the DIRECTED PDS problem of $H = (V, E)$. We rely on the *nice tree decomposition* of Lemma 4.1 in linear time as we assume bounded *tree-width* embedding. Now we commence by describing the state of the bags in our dynamic programming algorithm.

Given a directed graph $H = (V, E)$, where $H \subset G^{'}$ has been constructed as a *nice tree decomposition* $\langle X_i | i \in I, T \rangle$ of H with *tree-width* k. Let T_i denotes the subtree of T rooted at T-node i, and $Y_i = (\bigcup_{j \in V(T_i)} X_j) \setminus X_i$. Also let D_i be the subgraph induced by the vertices in the bags of T_i, that is, $D_i = H[Y_i \cup X_i]$, and let $D_i^{'} = H[X_i]$.

4.4.1 Definition States

During a bottom-up process the dynamic programming algorithm computes for every bag X_i the possible *Valid Colouring* of the edges of the subgraph $D_i^{'}$ and stores the sizes of the *origins* of a *Valid Colouring*. To avoid *dependency cycles* in all the possible colourings for $D_i^{'}$ when reaching bag X_i, we store the state of the bag X_i, where a *Valid Colouring* of D_i are characterized by the bag states. A bag state (s) of a bag X_i is a combination of the following states:

a. **The State of an Edge:**

   We define two edge states $s(e)$ for the edges $e = (uv) \in E(H[X_i])$ assigning the colour to e in the *colouring* C:

   i. $s(e) = (uv)$: Red,

   ii. $s(e) = (vu)$: Blue.

b. **The State of a Vertex:**

   According to Definition 4.6, we define four vertex degree states $s^d(v)$ in a bag X_i for every vertex $v \in X_i$ showing the number of the in-coming red edges, denoted by $s^{d^-}(v)$, and the out-going red edges, denoted by $s^{d^+}(v)$, between v and Y_i:

i. $s(v) = 1$: There is exactly one in-coming red edge from Y_i, and **no** out-going red edge from v to Y_i, where $s^{d^-}(v) = 1$ and $s^{d^+}(v) = 0$.

ii. $s(v) = 2$: There is exactly one in-coming red edge from Y_i, and one out-going red edge from v to Y_i, where $s^{d^-}(v) = 1$ and $s^{d^+}(v) = 1$.

iii. $s(v) = 3$: There is **no** red edges between Y_i and v.

iv. $s(v) = 4$: There is **no** in-coming red edge from Y_i, and there are at least two out-going red edges from v to Y_i, where $s^{d^-}(v) = 0$ and $s^{d^+}(v) \geq 2$.

v. $s(v) = 5$: There is **no** in-coming red edge from Y_i, and exactly one out-going red edge from v to Y_i, where $s^{d^-}(v) = 0$ and $s^{d^+}(v) = 1$.

Note that according to Definition 4.6, a vertex in a *Valid Colouring* cannot have more than one in-coming red edge (i.e. $d^-(v) = 1$), and a vertex with $d^-(v) = 1$ cannot have more than one out-going red edge (i.e. $d^+(v) = 1$). Hence, the above list covers all possible cases that should be considered.

c. **The State of a Coloured Vertex on a Path**:

According to Definition 4.9, we define seven vertex states $s(v)$ in a bag X_i for every vertex $v \in X_i$ depending on the directions of in/out blue and red edges in order to concatenate *dependency paths* and detect *dependency cycles*. For a path $(u, v) \in X_i$ with $(u \neq v)$, the state of (u, v) denoted by $s(u, v)$ shows if (u, v) can be concatenated with the other path in order to obtain a *dependency path* from u to v in $H[Y_i \cup \{u, v\}]$ such that $s(u, v) \subseteq \{\mathcal{F}, \mathcal{M}, \mathcal{E}, \mathcal{W}, \mathcal{Z}, \mathcal{Q}, \mathcal{L}\}$, or there may exist *dependency cycles* where $(u = v)$; that is, $s(u, v) \subseteq \{\mathcal{E}, \mathcal{Q}, \mathcal{L}\}$. Note that if the state of a coloured vertex on a path $s(u, v) = \emptyset$ then it means that there is no *dependency path* from u to v in $H[Y_i \cup \{u, v\}]$. The vertex states $s(v)$ is defined as follows:

i. $s(v) = \mathcal{F}$: If there is **no** in-coming edge and exactly one out-going red edge.

ii. $s(v) = \mathcal{M}$: If there is exactly one in-coming red edge and **no** out-going edge.

iii. $s(v) = \mathcal{E}$: If there is exactly one in-coming red edge and exactly one out-going red edge.

iv. $s(v) = \mathcal{W}$: If there is **no** in-coming edge and exactly one out-going blue edge.

v. $s(v) = \mathcal{Z}$: If there is exactly one in-coming blue edge and **no** out-going edge.

vi. $s(v) = \mathcal{Q}$: If there is exactly one in-coming blue edge and exactly one out-going blue edge.

vii. $s(v) = \mathcal{L}$: if there is exactly one in-coming red edge and exactly one out-going blue edge or vice versa.

As a consequence, for a bag X_i with $|X_i| \leq (k+1)$, we have at most $(7^{k+1} \cdot 5^{k+1} \cdot 2^{k+1})$ bag states; since the number of relevant edges, vertices, and coloured vertices on a Path are less than equal to $(k+1)$, $(k+1)$, $(k+1)$, respectively. We say that C is under the restriction of a bag state s of the bag X_i if C satisfies the following conditions:

1. The colouring of an edge $e \in E(H[X_i])$ coincides with a state given by $s(e)$.
2. All coloured vertices on a path $(u, v) \in X_i$, the type of the *dependency paths* from u to v in $H[Y_i \cup \{u, v\}]$ in the *colouring* C coincides with $s(u, v)$.
3. For each vertex $u \in X_i$, the number of red edges in the *colouring* C between u and Y_i coincides with $s^{d^-}(u)$ and $s^{d^+}(u)$.

4.4.2 A Concatenation Function of Dependency Paths

Let p_{ux} be the first *dependency path* where u and x are the tail and the head endpoint, respectively, and p_{xv} is the second *dependency path* with x and v as the tail and the head endpoint path, respectively. The task of the function is to take two given paths and concatenate them depending on $e^-_{p_{ux}}(x)$ (i.e. an in-coming edge to x) and $e^+_{p_{xv}}(x)$ (i.e. an out-going edge from x) in order to obtain the resulting path $p_{uv} = p_{ux} \otimes p_{xv}$ as follows:

- If $(u \neq v)$ then
  a. Take the edge $e^-_{p_{ux}}(x)$ and the edge $e^+_{p_{xv}}(x)$ and concatenate them.
  b. Check the obtained colour; that is $x \in \{\mathcal{E}, \mathcal{Q}, \mathcal{L}\}$.
  c. If a vertex x is coloured with $\mathcal{Q}$, then the obtained path is no longer a *dependency path*.
  d. Otherwise, the new *dependency path* p_{uv} is obtained by colouring a vertex x either $\{\mathcal{E}\}$ or $\{\mathcal{L}\}$ depending on in/out blue-red edges.

- Else if ($u = v$) then
  a. Concatenate the head endpoint of p_{ux} (i.e. $e^{-}_{p_{ux}}(x)$) and the tail endpoint of p_{xv} (i.e. $e^{+}_{p_{xv}}(x)$) such that $x \in \{\mathcal{E}, \mathcal{Q}, \mathcal{L}\}$.
  b. If a vertex x is coloured with $\mathcal{Q}$, then the obtained path is no longer a *dependency cycle*.
  c. Otherwise, the obtained path p_{uv} is a *dependency cycle* by colouring a vertex x whether $\{\mathcal{E}\}$ or $\{\mathcal{L}\}$ depending on in/out blue-red edges.

The crucial point in the dynamic programming is to check the degree constraints satisfying the properties in Definition 4.6 and detect *dependency cycles* in all possible colourings for D_i going through bag X_i.

4.4.3 Degree Constraints and Detecting Dependency Cycles

We use a procedure called valid (X_i, s) deciding if a *colouring* C of D_i under the restriction of (s) satisfies the degree constraints in X_i and also has no *dependency cycles* when reaching bag X_i.

a. **Degree Constraints**:

We denote $s^{-}(u)$ and $s^{+}(u)$ the number of the in-coming and out-going red edges (respectively) to/from u in the *colouring* of $H[X_i]$ given by the state of edges $s(e)$. The total number of the in-coming red edges to a vertex u, denoted by α_{in}, in the *colouring* C of D_i under the restriction of (s) is given by

$$\alpha_{in} = \Big(s^{-}(u) + s^{d^{-}}(u)\Big)$$

where $s^{d^{-}}(u)$ denotes the number of the in-coming red edges from Y_i to u, whereas the total number of out-going red edges $s^{+}(u)$ from a vertex u, denoted by α_{out}, in the *colouring* C of D_i under the restriction of s is given by

$$\alpha_{out} = \Big(s^{+}(u) + s^{d^{+}}(u)\Big)$$

where $s^{d^{+}}(u)$ denotes the number of out-going red edges from u to Y_i. Now we check the degree constraints at vertex $u \in X_i$ according to Definition 4.6 such that each $u \in PDS$ either:

i. $\Big((\alpha_{in} = 0) \wedge (\alpha_{out} \geq 2)\Big) \Longrightarrow \alpha_{out} \geq 2$, or
ii. $\Big((\alpha_{in} = 0) \wedge (\alpha_{out} \leq 1)\Big) \Longrightarrow \alpha_{out} \leq 1$

Other vertices in H (i.e. $u \notin PDS$):

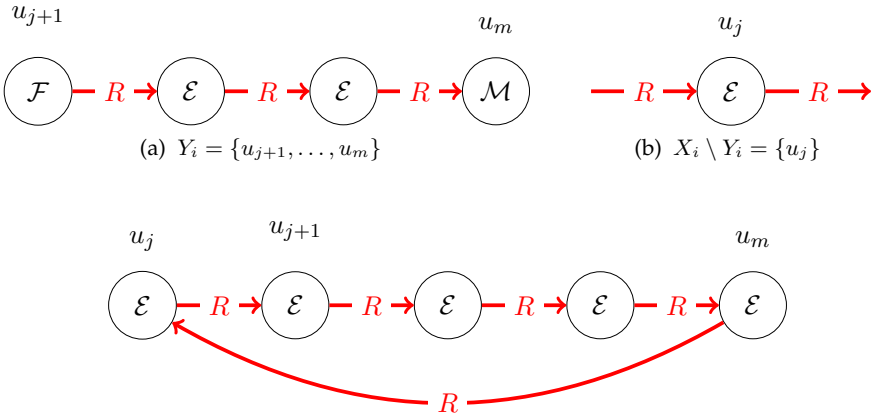
i. $\forall u \notin (V_D \cup V_S) \Longrightarrow \alpha_{\text{in}} \leq 1, \wedge/\vee$

ii. if $(\alpha_{in} = 1) \Longrightarrow \alpha_{out} = 1$.

b. **Dependency Cycles**:

Lemma 4.5 (The Time Complexity of a Dependency Cycle)

Given a dependency cycle $\mathcal{C}$ *with length* k *and the colours of all vertices in the* dependency cycle. *In the worst case, one can decide the* dependency path *is a cycle in time* $\mathcal{O}(k)$.



(a) $Y_i = \{u_{j+1}, \ldots, u_m\}$ (b) $X_i \setminus Y_i = \{u_j\}$

(c) As u_j has an in-coming and out-going red edge. After applying the concatenation function to Y_i with u_j, u_j should be coloured by $\mathcal{E}$, producing a dependency cycle.

Figure 4.5: The Detection of a *Dependency Cycle*

Proof. Let $\mathcal{C}$ be a *dependency cycle* $u_1, u_2 \ldots, u_m$ in D_i passing through some vertices in Y_i, where $Y_i = \{u_{j+1}, \ldots, u_m\}$ (see Subfig 4.5.(a)), and $X_i \setminus Y_i = \{u_j\}$ (see Subfig 4.5.(b)). Note that every two consecutive vertices in Y_i is connected such that the direction is going from u_{j+1} to u_m. Supposing that a vertex u_j has an in-coming red edge that is incident from u_m to u_j and has an out-going red edge from u_j to u_{j+1}; now the *dependency cycle* is built as shown in Subfig 4.5.(c). According to Definition 4.10, a cycle is formed by a sequence of directed edges such that all the red edges are in one direction, and there is no vertex with the colour $\mathcal{Q}$ (i.e. no two consecutive blue edges). As the colour of vertex u_j and the colour of vertices in the dependency paths from u_{j+1} to j_m in $H[Y_i]$ are all stored in the

bag state s. By applying the *colouring* vertices for each consecutive vertices in $H[Y_i \cup X_i]$ based on in/out blue-red edges, one can check if there is a *dependency cycle* going through vertices in Y_i when applying the concatenation function by searching for a vertex with only the colour $\mathcal{Q}$. Hence, a worst-case scenario is to visit all vertices in X_i, where $|X_i| \leq (k+1)$ and the number of *dependency cycles* going through X_i is $\mathcal{O}((k+1)!)$. Therefore, any one of the possible *dependency cycles* can be verified in with $\mathcal{O}(k)$ time.

4.5 Dynamic Programming Algorithm

Let Δ_i denotes the set of all possible states for the bag X_i. In a bottom-up dynamic programming strategy, we use a mapping $A_i : \Delta_i \to \mathbb{N} \cup \{+\infty\}$ for each bag X_i, where a bag state $s \in \Delta_i$ is the minimum size $A_i(s)$ of the *origins* in an optimal *Valid Colouring* C of all possible colourings of D_i under the restriction of s_i containing the state of vertex degrees (i.e. incoming and outgoing red edges), and coloured vertices in X_i.

4.5.1 The Initialisation Step

Our algorithm is initialised by setting a mapping A_i for each leaf node i of T. For each bag state $s_i \in \Delta_i$ we set $A_i(s_i) := \{+\infty\}$ if
$\Big(\exists v \in X_i : s_i^{d^-}(v) + s_i^{d^+}(v) \neq 0\Big) \vee \Big(\exists uv : (u \in X_i) \wedge (v \in X_i) \wedge (s_i(u,v) \neq \emptyset)\Big)$
$\vee \Big((\top[X_i], s_i) = \bot\Big)$
where there is directed edge between v and vertices in Y_i or there is a *dependency path* from u to v in $Di[Y_i \cup \{u,v\}]$ or if a *Valid Colouring* is false
Otherwise we define $A_i(s_i)$ as the number of vertices with no in-coming red edges in the *colouring* given by s_i:

$$A_i(s_i) := |\{vu \in X_i : \exists e = \{vu\} \in E(H[X_i]) \Longrightarrow s_i(e) = vu\}|$$

Hence only those bag states are taken into consideration where the edge states a *Valid Colouring*

4.5.2 The Bottom-up Computation Step

Now we perform a bottom-up step at each bag X_i visiting bags of the decomposition for computing the corresponding mapping A_i to X_i, and combining colourings (treating **Forget**, **Introduce**, **Join** nodes, and ultimately the **Root** node separately) satisfying:

1. Each forget node where $X_j = X_i \cup \{x\}$, we compute a bag state s_j of a bag X_j that is compatible with a bag state s_i of X_i.
2. For introduce nodes where X_i contains more edges than X_j such that $X_i = X_j \cup \{x\}$, we check whether a *Valid Colouring* of X_j under the restriction of s_j can be extended to a *Valid Colouring* for X_i under the restriction of s_i.
3. For join nodes i with two children (X_j,X_k), we check whether a *Valid Colouring* of X_j and X_k are combined with a *Valid Colouring* of X_i.

4.5.3 The Formal Definition of Compatibility

Now we define the formal definition of compatibility for each type of node separately (**Forget**, **Introduce**, **Join** nodes):

4.5.3.1 Forget Node

Definition 4.11 (The Computation of Forget Nodes in a *Valid Colouring*)
Supposing that node i is a forget node with child node j, where $X_j = X_i \cup \{x\}$, both D_i and D_j have the same set of vertices, so each Valid Colouring *of D_j is also a* Valid Colouring *of D_i. If there exists an edge between a vertex $v \in X_i$, and a vertex $x \in X_j$, then the state of degrees of vertices and coloured vertices can be different between $s_i^d(v)$ and $s_j^d(v)$. So, we say that s_j is compatible with s_i if it satisfies:*

1. $\forall v \in X_i$: *if* $\{xv\} \in E(H[X_j])$ *then* [1]
   a. $s_i^{d^-}(v) = 1$ *such that* $s_j(\{x, v\}) = (xv)$*; otherwise* $s_i^{d^-}(v) = s_j^{d^-}(v)$.
   b. $s_i^{d^+}(v) \geq 2$, *where* $s_i^{d^+}(v) = s_j^{d^+}(v) + 1$ *such that* $s_j(\{x, v\}) = (vx)$.
   c. *Otherwise* $s_i^{d^+}(v) \geq 2$ *such that* $s_i^{d^+}(v) = s_j^{d^+}(v)$.

[1] a,b and c (respectively) are illustrated in Figure 4.6

2. *If* $\{xv\} \notin E(H[X_j])$ *then* $\left(s_i^{d^-}(v) = s_j^{d^-}(v)\right) \wedge \left(s_i^{d^+}(v) = s_j^{d^+}(v)\right)$.
3. *For each edge* $e \in E(H[X_i])$, $s_i(e) = s_j(e)$.
4. *For each pair* $(u, v) \in X_i$, *we define the compatibility of* s_j *with respect to the state of coloured vertices on a path by applying the concatenation function of path such that*

$$s_i(u, v) = s_j(u, x) \otimes s_j(x, v).$$

Let (s_i) be the set of all bag states s_j for X_j that are compatible with the bag state s_i for X_i. The function A_i is computed for each bag state s_i as:

$$A_i(s_i) = \min_{s_j \in \Delta(s_i)} A_j(s_j)$$

As D_i and D_j have the same set of vertices, a *Valid Colouring* under the restriction of s_j is also a *Valid Colouring* under the restriction of s_i. Therefore, the computation of A_i is correct. However, if $\Delta s_i = \emptyset$ then, $A_i(s_i) = +\infty$.

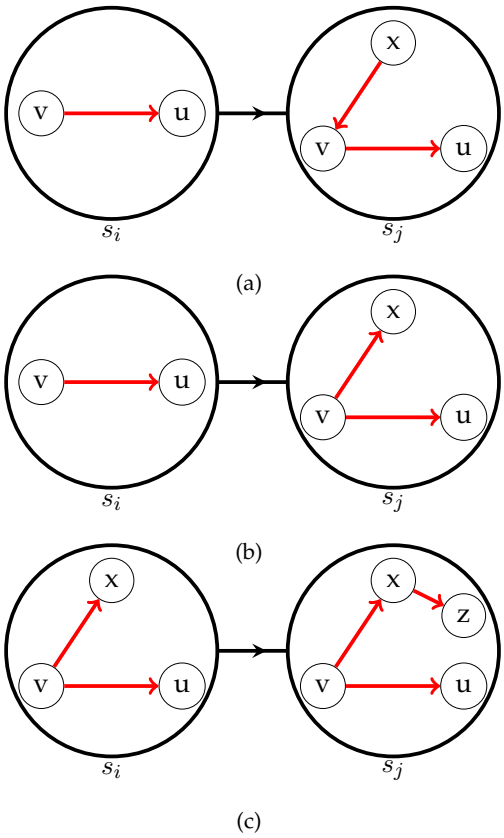


Figure 4.6: The Computation of Forget Node in a *Valid Colouring*

4.5.3.2 Introduce Node

Definition 4.12 (The Computation of Introduce Nodes in a *Valid Colouring*) *Supposing that a node i is an introduce node with a child j, where $X_i = X_j \cup \{x\}$. In order to compute $A_i(s_i)$ for a bag state s_i of the bag X_i, we compute the set of all bag states $\Delta(s_i)$ of the node j that are compatible with s_i. Note that the introduced vertex $x \in X_i$ has no neighbour in Y_i (due to the consistency property of* tree decompositions*), so the vertex x does not change the number of red edges between a vertex $v \in X_j$ and the vertices in Y_i. Therefore, s_j is compatible with s_i, if a* Valid Colouring *C under the restriction of s_i is a* Valid Colouring *under the restriction of s_j.*

Now we apply a procedure valid (X_i, s) deciding if there is an edge between a vertex $x \in X_i$ and the vertices in Y_i, such that $(X_i, s) = \perp$ if:

$x \in X_i : \Big(s_i^{d^-}(x) + s_i^{d^+}(x) \neq 0\Big) \vee \Big(\exists v \in X_j : (s_i(v,x) \neq \emptyset) \wedge (s_i(x,v) \neq \emptyset)\Big)$

Otherwise (Δs_i) contains a bag state s_j of X_j if:

$\forall e \in E(H[X_j]) : \Big(s_i(e) = s_j(e)\Big) \wedge \Big(\forall v \in X_j : s_i^d(v) = s_j^d(v)\Big) \wedge$

$\Big(\forall uv \in X_j : s_i(u,v) = s_j(u,v)\Big)$

Note due to a new introduced vertex x, the degree constraints for a *Valid Colouring* can be violated by a vertex in $X_i \setminus \{x\}$ (i.e. the edges incident to x in $H[X_i]$). Hence, we should verify if there exists a *dependency cycle* passing through x.

a. **Degree Constraints:** as $X_i \setminus \{x\} = X_j$, all in/out red edges e in $E(D_j)$ incident to v can be verified by the information stored in $s_j(v)$ and $s_j(e)$ under the restriction of s_j. Together with a given edge $e = \{v, x\}$ by $s_i(e)$, one can check the in/out red edges of v in the *colouring* of D_i under the restriction $s_i(e)$ in $\mathcal{O}(|(k+1)^2|)$ time as the number of edges in each bag has at most $(|k+1|)$ edges without self-loop.

b. ***Dependency Cycles*:** by applying the *7-Colourings* vertices to a path with $u, v \in X_j$, one can check if there is a *dependency cycle* passing through x in the *colouring* of D_i under the restriction of s_i, if there is no vertex with the colour $\mathcal{Q}$.

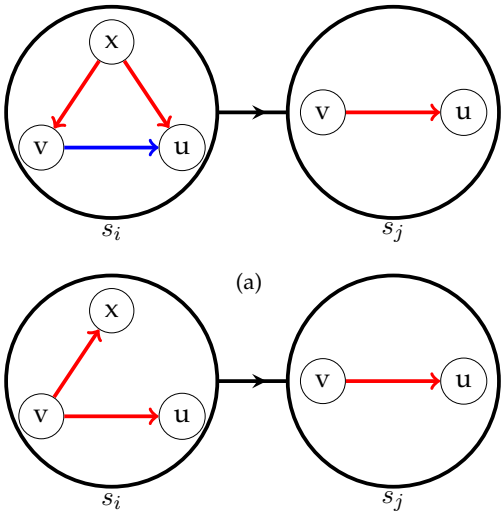


Figure 4.7: The Computation of Origins of Introduce Node in a *Valid Colouring*

Now we compute an *origin* in $A_i(s_i)$ for a bag state s_i as follows:

Definition 4.13 (The Computation of Origins of Introduce Nodes in a *Valid Colouring*)

- ***First Case:*** *a new introduced vertex x has edges that are incident from x to $v \in X_i$, then x is counted as an* origin, *where some of the* origins *in s_j might not be* origins *in s_i (see Subfig 4.7.(a)). Hence, the mapping A_i for X_i is computed as:*

$$A_i(s_i) = \min_{s_j \in \Delta(s_i)} \{A_j(s_j) + |\Lambda_{s_i}(x)|\}$$

where $\Lambda_{s_i}(x)$ denotes an introduced vertex x as an origin (i.e. x has at least two edges $e = \{xu\}$ and $e^{'} = \{xv\}$ in $D[X_i]$ that are incident from x to the vertices in $X_i \setminus \{x\}$ with $s_i(e) = xu$ and $s_i(e^{'}) = xv$.

- ***Second Case:*** *the* origins *in s_j are also* origins *in s_i such that $s_i^{d+}(v) = 2$ where there exists an edge $e = \{vx\}$ in $D[X_i]$ with $s_i(e) = vx$ and the other edge $e^{'} = \{vu\}$ in $D[X_j]$ with $s_j(e^{'}) = vu$ in a* Valid Colouring *of D_j under the restriction of s_j (see Subfig 4.7.(b)). Thus, the mapping A_i for X_i is computed as follows:*

$$A_i(s_i) = \min_{s_j \in \Delta(s_i)} \{A_j(s_j) + |\beta_i(s_j)|\}$$

where $\beta_i(s_j)$ denotes a vertex v as an origin (i.e. v has at least two edges $e = \{vx\}$ and $e^{'} = \{vu\}$ in $D[X_i]$ that are incident from v to the vertices in X_i with $s_i(e) = vx$ and $s_i(e^{'}) = vu$.

4.5.3.3 Join Node

Definition 4.14 (The Computation of Join Nodes in a *Valid Colouring*)

Supposing that i is a join node with children j and l, where $X_i = X_j = X_l$. Let $\Delta(s_i)$ be the set of the bag state pairs (s_j, s_l) that are compatible with s_i, where s_j and s_l denote bag states of X_j and X_l, respectively. Note that, due to the properties of tree decompositions, *$Y_j \cap Y_l = \emptyset$, and thus we have $Y_i = Y_j \cup Y_l$. The set, $\Delta(s_i)$, is compatible with s_j and s_l with respect to vertex $v \in X_i$ if the following conditions are satisfied:*

1. *The sum of the in-coming edges of v in $H[Y_j \cup \{v\}]$ and in $H[Y_l \cup \{v\}]$ is equal to the in-coming edges of v in $H[Y_i \cup \{v\}]$, and the sum of the out-going edges of v in $H[Y_j \cup \{v\}]$ and in $H[Y_l \cup \{v\}]$ is equal to the out-going edges of v in $H[Y_i \cup \{v\}]$ such that there are at least two red edges from a vertex $v \in X_i$ to a vertex in Y_j and Y_l, respectively. Hence $v \in X_i$:*
   a. $s_i^{d^-}(v) = \Big(s_j^{d^-}(v) + s_l^{d^-}(v)\Big)$.
   b. $s_i^{d^+}(v) \geq 2$ *such that* $s_i^{d^+}(v) = \Big(s_j^{d^+}(v) + s_l^{d^+}(v)\Big)$.
2. *The edge state of $s_j(e)$ and $s_l(e)$ are compatible with the edge state given by $s_i(e)$ for each edge $e \in E(H[X_i])$ such that*
$$s_i(e) = s_j(e) = s_l(e)$$
3. *By applying the* 7-Colourings *vertices to a path $(u, v) \in X_i$, s_i is compatible with s_j and s_l such that the concatenation of both* dependency path *are satisfied:*
$$s_i(u, v) = \Big(s_j(u, v) \cup s_l(u, v)\Big)$$

Note that the combination of the two colourings of D_j and D_l, which are under the restrictions of s_j and s_l, respectively, should be coincided with a *Valid Colouring* of D_i under the restriction of s_i. Therefore, the combination of D_j and D_l results in new *origins* in a *Valid Colouring* of D_i (see Figure 4.8), where either:

a. A vertex $v \in X_i$ that is an origin in a *Valid Colouring* of D_j or D_l may not be an origin in D_i, or

b. a vertex $v \in X_i$ that is an origin in both *Valid Colourings* of D_j and D_l may be counted twice in D_i.

With respect to the two conditions (a,b), we now compute the mapping $A_i(s_i)$ for a bag state s_i:

$$A_i(s_i) = \min_{(s_j,s_l)\in\Delta(s_i)} \{A_j(s_j) + A_l(s_l) + |\Lambda_{s_i}(s_j,s_l)| - |(\beta_{si}(s_j,s_l)|)\}$$

where the set $\Lambda_{s_i}(s_j,s_l)$ contains the vertices $v \in X_i$ which satisfy the following condition:

- A vertex $v \in X_i$ that is an origin in both D_j and D_l, and there is no edge $e = uv \in E(H[X_i])$ with $s_i(e) = uv$, such that
  $v \in X_i : \Big((s_j^{d^-}(v) = 0) \wedge (s_j^{d^+}(v) = 1)\Big) \wedge \Big((s_l^{d^-}(v) = 0) \wedge (s_l^{d^+}(v) = 1)\Big) \wedge \Big((e = \{uv\} \notin E(H[X_i]) : s_i(e) = uv)\Big)$

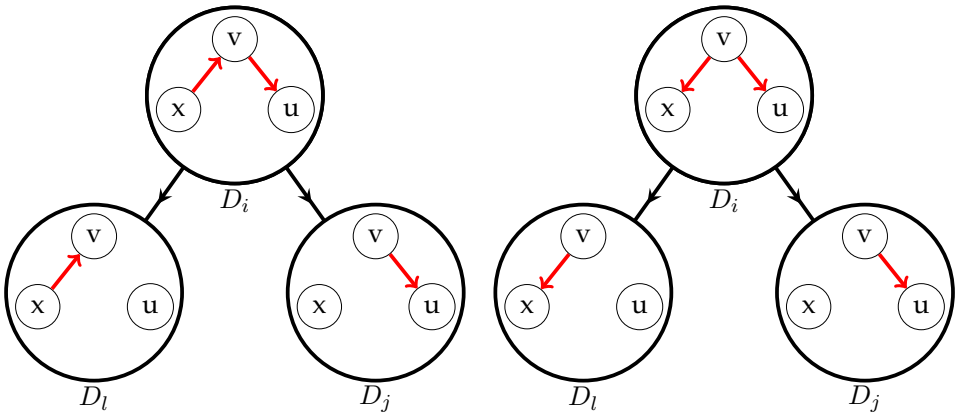


Figure 4.8: The Computation of Origins of Join Node in a *Valid Colouring*

Whereas the set $\beta_{si}(s_j,s_l)$ contains the vertices $v \in X_i$ which satisfy at least one of the following conditions:

a. A vertex $v \in X_i$ that is an origin in D_j **but not** in D_l may not be an origin in D_i (see Subfig 4.8.(a)), such that
   $v \in X_i : \Big((s_j^{d^-}(v) = 0) \wedge (s_j^{d^+}(v) = 1)\Big) \wedge \Big(e = \{xv\} \in E(H[X_i]) : s_i(e) = xv\Big)$

b. A vertex $v \in X_i$ that is an origin in D_l **but not** in D_j may not be an origin in D_i, such that
   $\Big((s_l^{d^-}(v) = 0) \wedge (s_l^{d^+}(v) = 1)\Big) \wedge \Big(e = \{xv\} \in E(H[X_i]) : s_i(e) = xv\Big)$

c. A vertex $v \in X_i$ that is an origin in both D_j and D_l may be counted twice in D_i (see Subfig 4.8.(b)), such that

$$v \in X_i : s_i^{d^-}(v) + s_i^-(v) = 0$$

where $s_i^-(v)$ denotes the number of the in-coming red edges that are incident to v in $H[X_i]$, and $s_i^{d^-}(v)$ the number of the in-coming red edges that are incident from Y_i to v.

4.5.3.4 The Computation of Root

Definition 4.15 (The Computation of Root r in a *Valid Colouring*)

As the dynamic programming algorithm is a bottom-up strategy. Now we compute the minimum origins *in a* Valid Colouring *of H at the root r of the* tree decomposition *T:*

$$\delta(H) = \min_{s \in \Delta_r} \{A_r(s)\}$$

We give this result also in constructive form in the following algorithm:

Algorithm 4.1: The Generation of a PDS for a Directed Graph $H = (V, E)$

Input : Given a nice tree decomposition $\langle X_i | i \in I, T \rangle$ of $H = (V, E)$, where the underlying undirected graph has bounded *tree-width*

Output: A set of origins (S) for H, where $S \in PDS$

$S \leftarrow \emptyset$;
Let d be the maximum distance from the root $r \in T$;
Let $A_i(s_i)$ be a mapping for each bag X_i;
forall the *leaves nodes* X_i *of* T **do**
    if $(X_i, s_i) = \perp$ **then**
        $A_i(s_i) \leftarrow \{+\infty\}$;
    else
        • Apply *Valid Colouring* (X_i, s_i), and compute
        $A_i(s_i) := \{vu \in X_i : \exists e = \{v, u\} \in E(H[X_i]) \Longrightarrow s_i(e) = vu\}$;
for *i= d to 0* **do**
    if X_i *is a Forget node* **then**
        • Compute all s_j for X_j that are compatible with s_i;
        • Set $A_i(s_i) = \min_{s_j \in \Delta(s_i)} A_j(s_j)$;
    else if X_i *is an Introduce node* **then**
        • Check a new introduced vertex $s_i^d(x)$;
        • Check Dependency Cycles passing through x;
        • Compute A set of the node j that are compatible with s_i;
        if x *is counted as an origin where x has edges that are incident to* $v \in Xi$, **then**
            $A_i(s_i) = \min_{s_j \in \Delta(s_i)} \{A_j(s_j) + |\Lambda_{s_i}(x)|\}$;
        else
            The origins in s_j are also origins in s_i:
            $A_i(s_i) = \min_{s_j \in \Delta(s_i)} \{A_j(s_j) + |\beta_i(s_j)|\}$;
    else
        • X_i is a Join node
        • Compute the set pairs (s_j, s_l) that are compatible with s_i;
        • Check the origin $v \in S$;
        if $\big(v \in D_j \wedge v \notin D_l\big) \vee \big(v \in D_l \wedge v \notin D_j\big) \vee \big(v \in D_j \wedge v \in D_l\big)$ **then**
            $A_i(s_i) = \min_{(s_j, s_l) \in \Delta(s_i)} \{A_j(s_j) + A_l(s_l) + |\Lambda_{s_i}(s_j, s_l)| - |\beta_{si}(s_j, s_l)|\}$;
Compute the minimum origins at r, $\delta(H) = \min_{s \in \Delta_r}, \{A_r(s)\}$;
$S \leftarrow \delta(H)$;
return S;

4.6 Time Complexity

Lemma 4.6 (Time Complexity of a PDS for a Directed Graph)

Given a tree decomposition *of width k for a directed graph $H = (V, E)$, where the underlying undirected graph has bounded* tree-width*, one can solve a PDS in $\mathcal{O}(nc^{k^2})$ time for a constant c.*

Proof. Let $H = (V, E)$ be a directed graph satisfying the same assumptions (1,2 and 3) as $G^{'}$ shown in Subsection 1.4.2 in Chapter 1, where the underlying undirected graph has bounded *tree-width*. The analysis of the presented algorithm, satisfying the assumption (4) in Subsection 1.4.2 in Chapter 1, in terms of time complexity can be classified as Best case, Worst case and Average case.

Assuming that there exists uncovered vertices denoted by W of H and the vertices of W are not included in **join** nodes. Thus, the resulting algorithm will have a best-case complexity of $\mathcal{O}(nc^k)$. Now, assuming that a PDS of H is completely different to the remainder of the original $G^{'}$. This requires the reconstruction of a new PDS for H from scratch and thus the resulting algorithm will have a worst case of $\mathcal{O}(nc^{k^2})$, where the most time-consuming part of the algorithm is to determine the mapping A_i at a join node i. Therefore, we compute all possible bag states once for each child bag such that there are at most $(7^{k+1} \cdot 5^{k+1} \cdot 2^{k+1})$ states for each bag X_i with $|X_i| \leq (k+1)$. Hence, there can be $(7^{(k+1)^2} \cdot 5^{(k+1)} \cdot 2^{(k+1)^2})$ states which are compatible to s_i.

For the time complexity of the average case, suppose given H has the remaining PDS of the original $G^{'}$. This means that the uncovered vertices of H are included in at most the half bags of tree $n/2$. Hence, there may exist some uncovered vertices in join nodes; if this is the case, then the most time-consuming part of the algorithm is to compute the mapping A_i at a join node i where the resulting algorithm will have an average complexity of the time of computation A_i at each bag X_i is $\mathcal{O}(\log nc^{k^2})$ for a constant c. However, if the vertices of W do not exist in join nodes, then the running time of the algorithm in the average case is $\mathcal{O}(\log nc^k)$. The summary of the time complexity of the algorithm is illustrated in the following table:

Case	Time Complexity	Given Input
Best-case	$\mathcal{O}(n\ c^k)$	The uncovered vertices $W \notin$ join nodes
Average-case	$\mathcal{O}(\log n\ c^{k^2})$	The remainder of a PDS in $G^{'}$ and $W \in n/2$
Worst-case	$\mathcal{O}(n\ c^{k^2})$	a PDS of H is different to a PDS of $G^{'}$

Table 4.1: The summary of time complexity of the Algorithm 4.1

4.7 Summary

The *resilience* of control structures and the rapid ability to recover *controllability* after compromise, and other attacks resulting particularly in partitionings of the original graphs is a significant problem in control systems. This also means that an adversary with sufficient knowledge of the network distribution and its power domination can disconnect parts of the control graph and leave parts of a system uncontrolled.

In this chapter, we proposed a reconstruction algorithm for (directed) control graphs of bounded *tree-width* embedded in Erdős-Rényi random graphs after an event or attack leading to a degradation of the control of the network and a significant reduction of its observability. We also presented the *7-Colourings* of a *dependency path* to determine the direction of edges and their colours to concatenate *dependency paths* and detect *dependency cycles* while computing bags in a *nice tree decomposition*. Our result reduces the number combinations by reducing c, where there are at most $(7^{(k+1)^2} \cdot 5^{k+1} \cdot 2^{(k+1)^2})$ states for each bag X_i with $|X_i| \leq (k+1)$, also reducing the time complexity required to compute a *Valid Colouring* at each bag owing to a reduction in the number of colours required. This also allows faster a re-construction of a PDS, and ultimately the re-gaining of control for operators of control systems.

Chapter 5

Updating a Power Dominating Set

5.1 Overview

As the underlying POWER DOMINATING SET problem does not permit efficient re-computation, this chapter proposes to reduce the average-case complexity of a reconstruction algorithm for (directed) control graphs proposed in the previous chapter by re-using remaining fragments of the original graph where possible and identifying previously un-used edges to minimise the number of a PDS. This algorithm relies on *Depth-First Search* approach (DFS) which yields improved average-case complexity over previous algorithm, where an average-case complexity of computing a PDS is $\mathcal{O}(|V + \Big\{E_t \cup E_f\big[N(v)\big] \cup E_c\big[N(v)\big]\Big\} \setminus E_b|)$ time, where $v \in PDS$.

5.2 Problem Statement and Assumption

The aim of this algorithm is to reduce the average-case complexity of a reconstruction algorithm proposed in Chapter 4. Recall that the problem statement in this chapter is exactly the same as the previous chapter, where $G^{'} = (V, E)$ is a directed graph, generated by $ER(n, p)$, such that $G^{'}$ satisfies the assumptions (1,2 and 3) as shown in Subsection 1.4.2 in Chapter 1.

Let $S^{'}$ be a given PDS of $G^{'}$, we assume that $G^{'} = (V, E)$ is subjected to intentional or random removal, where an adversary with knowledge of the network distribution and its power domination can disconnect parts the of control graph and leave parts of a system uncontrolled (*e.g.* when an attacker has physical access to the actuator and able to install its own actuators or corrupt all the actuator channels).

Therefore, the deletion of vertices of $G^{'}$ may result in a disconnected component of a directed graph $G^{'}$, defined by $H = (V, E)$, where $H = (V, E) \subset G^{'}$ is partitioned from the original graph $G^{'}$ such that $V(H) \notin V(G^{'})$ and $E(H) \notin E(G^{'})$ (i.e. there exists no edge in H whose one end vertex is in $G^{'}$ and verse vice). For the disconnected component H that satisfies the same assumptions (1,2 and 3) as $G^{'}$ mentioned in Subsection 1.4.2 in Chapter 1, we suppose that H has a remaining structures of the original graph $G^{'}$ and un-used edges. While designing the algorithm, we assume a complete view of the status of the graph after an attack, and suppose the computation time of the algorithm is related to time complexity and not real time.

The contribution of this chapter compared to the previous one is to enhance the time complexity of a reconstruction algorithm proposed in Chapter 4 after an event or attack leading to the partitioning of the original control network by re-using remaining structures of the original where possible to recover a control graph and identifying previously un-used edges to minimise a PDS for $H = (V, E)$.

•**Input:**

We are given a *Depth-First Search* (DFS) structure for a disconnected component $H = (V, E)$ of the original graph $G^{'}$ where $H = (V, E) \subset G^{'}$.

•**Question:**

Can we design an algorithm to enhanced the average-case complexity of a reconstruction algorithm in the previous chapter to compute a PDS for H, denoted by S, after the original graph has been modified (*e.g.* in real scenarios, when an attacker able to estimate the state of the system and corrupt the actuator channels by launch a false-data injection attack). A particular scenario of the reconstruction of the PDS problem in the context of electrical power network control is to reconstruct a minimum-sized set of measurement devices when the system under attack.

•**Output:**

A minimal $S \subseteq V$ for H by using *tree*, *forward* and *cross edges* in a DFS structure such that all vertices in V are controlled by the vertices in S.

5.3 Depth-First Search (DFS)

This algorithm assumes a *Depth First Search* for H is giving; we now define a DFS:

Definition 5.1 (Depth First Search (DFS), Gibbons [46])

DFS is a systematic method of visiting the vertices of a graph (i.e. a directed oran undirected graph). Its general step requires that if we are currently visiting vertex u, then we next visit a vertex adjacent to u that has not yet been visited. If no such vertex exists then we return to the vertex visited just before u and the search is repeated until every vertex in that component of the graph has been visited.

5.3.1 DFS Algorithm

The DFS procedure takes as input a graph G, and outputs its predecessor subgraph in the form of a depth-first forest. In addition, it assigns two timestamps to each vertex: discovery and finishing time. The algorithm initialises each vertex to "white" to indicate that they are not discovered yet. It also sets each parent of vertex to null. The procedure begins by selecting one vertex u from the graph, setting its color to "grey" to indicate that the vertex is now discovered (but not finished) and assigning to it discovery time 0. For each vertex v that belongs to the set $Adj[u]$, and is still marked as "white", DFS-Visit is called recursively, assigning to each vertex the appropriate discovery time $d[v]$ (the time variable is incremented at each step). If no white descendant of v exists, then v becomes black and is assigned the appropriate finishing time, and the algorithm returns to the exploration of the ancestor of a vertex $\pi(v)$. If all of u's descendants are black, u becomes black and if there are no other white vertices in the graph, the algorithm reaches a finishing state, otherwise a new "source" vertex is selected, from the remaining white vertices, and the procedure continues as before [70, 100].

The initialisation part of DFS has time complexity $\Theta(n)$, as every vertex must be visited once so as to mark it as "white". The main (recursive) part of the algorithm has time complexity $\Theta(m)$, as every edge must be crossed (twice) during the examination of the adjacent vertices of every vertex. In total, the time complexity of algorithm is $\Theta(n + m)$.

5.4 Reconstructing a Directed PDS via a DFS

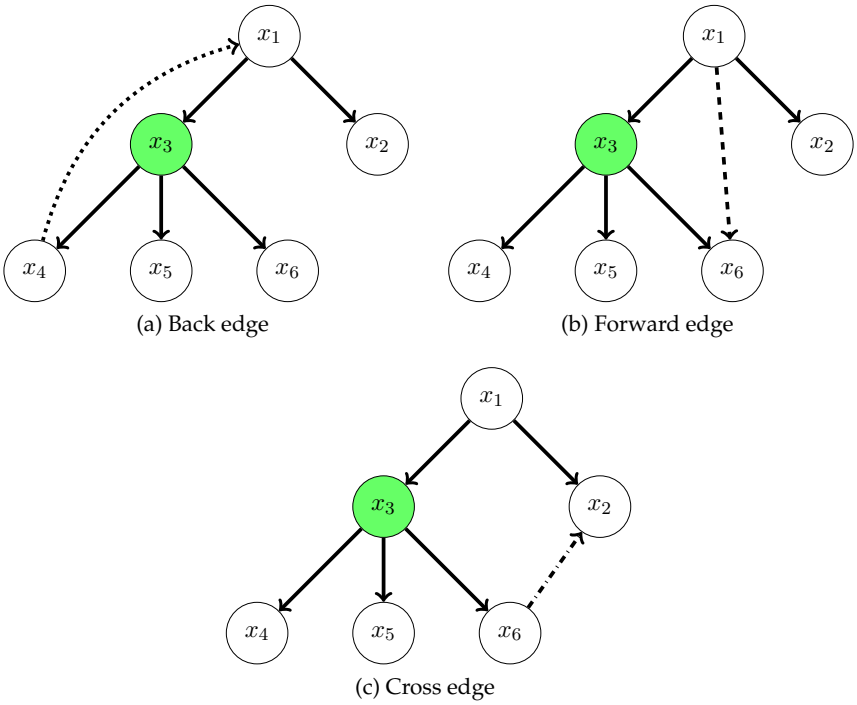
DFS can be used to seek edges of a given directed graph $H = (V, E)$, identifying *articulation points* and constructing the H as a tree-like structure, which gives equivalent characterisations of trees. DFS separates edges into groups revealing information on $H = (V, E)$, and *articulation points* or (*Cut-Vertices*) giving an equivalent formulation for identifying a PDS in undirected graphs as shown by [107] for block graphs. An *articulation point* or *Cut-Vertex* is a vertex whose removal (together with the removal of any incident edges) results in a disconnected graph. There are several well-known efficient algorithms for a construction based on DFS in $\mathcal{O}(V + E)$ time [70, 100]. We now can define four edge types, including *tree edge* (E_t), *forward edge* (E_f), *cross edge* (E_c) and *back edge* (E_b), produced by DFS on a given directed graph $H = (V, E)$ as follows:

Definition 5.2 (DFS Edge Classification)

During executing DFS on graph H, we select the existing PDS as the root to traverse an entire graph. Therefore, edges can be classified by type:

1. *If v is visited for the first time as we traverse the edge (uv), then the edge is* a tree edge *describing a relation between a vertex and one of its direct descendants.*
2. *If v has already been visited:*
   a. *If v is an ancestor of u, then edge (uv) is a* back edge *(i.e. connecting a vertex u to an ancestor v in DFS tree (self-loop), which may occur in directed graphs).*
   b. *If v is a descendant of u, then edge (uv) is a* forward edge *(i.e. non-*tree edges *(uv) connecting a vertex u to a descendant v in DFS tree).*
   c. *If v is neither an ancestor nor descendant of u, then edge (uv) a* cross edge*.*

When drawing diagrams, we represent edges $\{E_t, E_b, E_f, E_c\}$ by line types as shown in Figure 5.1, where a solid line (E_t) represents a *tree edge* ———, a dotted line (E_b) a *back edge* ....., a dashed line (E_f) a *forward edge* - - - - -, and a dash-dotted line (E_c) a *cross edge* -.-.-., respectively. Now we identify articulation points on a *Directed PDS* as follows:



(a) Back edge

(b) Forward edge

(c) Cross edge

Figure 5.1: Articulation Points with Different Edge Types in a DFS

Definition 5.3 (Cut-Vertex or Articulation Points in a DFS)

Given a directed graph $H = (V, E)$ constructed by a DFS, an articulation point *u is said to be in a PDS if it satisfies*

a. *The root r of the tree in DFS is an articulation if $d^+(r) \geq 2$ is connected by a* tree edge*, and no* cross edges *between the subtrees of the root.*

b. *Any other internal vertex v in the tree (other than the root) is an* articulation point *if it has a subtree rooted at a child of v such that there is no* back edge *from any vertex in this subtree connected to a higher level vertex than v and $d^+(v) \geq 2$.*

c. *A vertex v is in a PDS if $d^+(v) \geq 2$ connected by a* tree edge*, and there exists a* cross edge *that is incident from the subtrees of v to neither an ancestor or descendant of v (or inversely), or a* back edge *connected to a higher level vertex than v,*

d. *If a vertex v has only one child (i.e. $d^+(v) \geq 1$ and $d^-(v) = 0$), then $v \in PDS$.*

e. *A leaf vertex is not an* articulation point *as its removal from a tree does not affect the remainder of the tree, thus the tree remains connected.*

Theorem 5.1 (Cut-Vertex or an Articulation Point in a PDS)

Given a DFS for a directed graph $H = (V, E)$, $v \in PDS$ is said to be an articulation point *(or* Cut-Vertex*) if it satisfies*

1. *v has more than one child connected through a* tree edge *where $d^+_{E_t}(v) \geq 2$.*
2. *There is no* back edge *that is incident from any child x in the subtrees of v such that x is connected to a higher level vertex than v (see Subfig 5.1.(a)).*
3. *There is no* forward edge *that is incident from an ancestor of v to any child in the subtrees of v (see Subfig 5.1.(b)).*
4. *There is no* cross edge *that is incident from the subtrees of v to neither an ancestor or descendant of v (or vice versa) (see Subfig 5.1.(c)).*

Proof. Let $H = (V, E)$ be a directed graph, satisfying the same assumptions (1,2 and 3) as G' shown in Subsection 1.4.2 in Chapter 1. Assume a DFS structure for H is given where it contains $\{x_1, x_2, \ldots, x_n\}$ rooted at x_1, and supposing that there exists a vertex x_3 with more than one child, where $d^+(x_3) \geq 2$, and one of its children is connected to the ancestor of x_3 by a *back edge* (x_4, x_1) (see Subfig 5.1.(a)). The Proof is by contradiction in three cases (we use Figure 5.1 to demonstrate the construction):

1. If one of the subtrees (here: x_4) of x_3 are connected to an ancestor of x_3 by a *back edge* (see Subfig 5.1.(a)), then a vertex x_4 is still connected to an ancestor of x_3 by a *back edge* after omitting x_3. Therefore, x_3 is not an *articulation point*.
2. If there exists a vertex x_6 of the subtrees of x_3 connected to the ancestor of x_3 by a *forward edge* (see Subfig 5.1.(b)), then x_6 is still connected to H after omitting x_3, thus x_3 is not an *articulation point*.
3. If there exists a vertex x_6 of the subtrees of x_3 connected to neither an ancestor or descendant of x_3 by a *cross edge* (see Subfig 5.1.(c)), then after removing x_3, x_6 will be connected to H, thus x_3 is not an *articulation point*.

We can now reformulate a *colouring* of a directed graph $H = (V, E)$ constructed by a DFS. This reformulation is based on a *Valid Colouring* as defined in Definition 4.6 in Chapter 4. Our approach applies to a directed graph $H = (V, E)$ such that H is structured by a DFS:

Definition 5.4 (A Colouring of a Directed Graph in a DFS)

Given a DFS for a directed graph $H = (V, E)$, a colouring *of $H = (V, E)$ is a partition of the edges of H into red and blue edges. We denote the* colouring *by $C = (V, E^r)$ where E^r is the set of red edges.*

Before introducing a *Valid Colouring* of a directed graph in a DFS structure, we now define an origin of a *Valid Colouring* in a DFS:

Definition 5.5 (The Origins of a *Valid Colouring* in a DFS)

Given a DFS for $H = (V, E)$, we refer to a vertex v as an origin of the colouring *of H in a DFS if it satisfies:*

1. *It is an* articulation point *(or a* Cut-Vertex*).*
2. *It has no in-edge in H, where $v \in H : d^-(v) = 0$.*
3. *It has no red in-edges in H^r such that $v \in H^r : d^-(v) = 0$, where H^r denotes the set of vertices with red edges.*

Definition 5.6 (A *Valid Colouring* of a Directed Graph in a DFS)

A Valid Colouring *of edges of $H = (V, E)$ is a* colouring *of H into red and blue edges satisfying:*

1. *An origin of the* colouring *in $H^r = (V, E^r)$ satisfies:*
   a. *The root of a DFS may exist a* Cut-Vertex*, denoted by $(v_{R_{ct}})$, with out-degree of at least 2 and no in-edge incident to $(v_{R_{ct}})$ with **no** cross edge between the subtrees of $(v_{R_{ct}})$, **has** at least 2 out-red edges:*

   $$\exists v \in V(H) : \Big(\big(d^-_H(v_{R_{ct}}) = 0\big) \wedge \big(d^+_H(v_{R_{ct}}) \geq 2\big) \wedge \big(\forall\ child\ x\ of\ (v_{R_{ct}}) : E_c(x) = \emptyset\big)\Big) \Longrightarrow d^+_{H^r}(v_{R_{ct}}) \geq 2$$

   b. *The root of a DFS, denoted by v_R, with out-degree of at least 2 and no in-edge incident to v_R and there is at least one* cross edge *between the subtrees of v_R, **has** at least 2 out-red edges (i.e. v_R is not* Cut-Vertex*):*

   $$\exists v \in V(H) : \Big(\big(d^-_H(v_R) = 0\big) \wedge \big(d^+_H(v_R) \geq 2\big) \wedge \big(\exists\ child\ x\ of\ v_R : E_c(x) \geq 1\big)\Big) \Longrightarrow d^+_{H^r}(v_R) \geq 2$$

   c. *There may exist a* Cut-Vertex*, denoted by v_{ct}, with out-degree of at least 2 and no in-red edge incident to $E_{H^r}(v_{ct})$ and no* cross edges *between the subtrees of v_{ct} and no* back edge *incident to an ancestor of v_{ct}, **has** at least 2 out-red edges:*

$$\exists v \in V(H) : \Big(\big(d^-_{H^r}(v_{ct}) = 0\big)\wedge\big(d^+_H(v_{ct}) \geq 2\big)\wedge\big(\forall\ child\ x\ of\ v_{ct} : E_c(x) = \emptyset \wedge E_b(x) = 0\big)\Big) \Longrightarrow d^+_{H^r}(v_{ct}) \geq 2$$

d. There may exist a domination vertex, denoted by v_D, with out-degree of at least 2 and no in-red edge incident to $E_{H^r}(v_D)$ and there is at least one cross edge *between the subtrees of v_D or* back edge *incident to an ancestor of v_D, **has** at least 2 out-red edges (i.e. V_D is no* Cut-Vertex*):*

$$\exists v \in V(H) : \Big(\big(d^-_{H^r}(v_D) = 0\big)\wedge\big(d^+_H(v_D) \geq 2\big)\wedge\big(\exists\ child\ x \in v_D : E_c(x) \geq 1 \vee E_b(x) geq 1\big)\Big) \Longrightarrow d^+_{H^r}(v_D) \geq 2$$

*e. There may exist a simple vertex, denoted by v_S, with no in-degree and at least out-degree of exactly one, **has** at least one out-red edge:*

$$\exists v \in V(H) : \Big(\big(d^-_H(v_S) = 0\big)\wedge\big(d^+_H(v_S) \geq 1\big)\Big) \Longrightarrow d^+_{H^r}(v_S) \geq 1$$

2. The remaining vertices in $\{H \setminus (v_{R_{ct}} \cup v_R \cup v_{ct} \cup v_D \cup v_S)\}$ covered by the red edges in $H^r = (V, E^r)$ have the following properties:

i. $\forall v \in H : d^-_{H^r}(v) \leq 1$, and

ii. $\forall v \in H : d^-_{H^r}(v) = 1 \Longrightarrow d^+_{H^r}(v) \leq 1$.

*3. H has **no*** dependency cycle*. A* dependency cycle *is a sequence of directed edges whose underlying undirected graph forms a cycle such that all the red edges are in one direction, all the blue edges are in the other direction, and there are no two consecutive blue edges.*

Note a vertex with $d^-_{H^r}(v) = 0$ in $H^r = (V, E^r)$ is an origin, denoted by Φ, of C.

Definition 5.7 (A Dependency Path in a *Valid Colouring* of a DFS)
Given a DFS for $H = (V, E)$ and a Valid Colouring *of H, a* dependency path *in a* Valid Colouring *is a sequence of red edges, where $P = v_1, e_1, v_2, e_2, \ldots, e_{i-1}, v_i$, such that P has no* back edge *coloured red (i.e. all red edges are directed away from the start vertex (v_1) of P and ends with a vertex (v_i)). The length of a* dependency path *is defined as the number of red edges in the path. A* dependency cycle *in a DFS of a directed graph is a sequence of directed edges whose underlying undirected graph forms a cycle such that all the red edges are in one direction, and a red* back edge *in the other direction.*

We can now define the *colouring* of neighbours vertices of a PDS:

Definition 5.8 (Colouring of the Neighbours of a PDS in a DFS)

In order to obtain a minimal PDS in H, we define two colours for $N(v)$ where $v \in PDS$ depending on forward *and* cross edges *which are incident from $u \in PDS$ to $N(v)$, such that a* colouring *of the neighbours of a PDS in $H = (V, E)$ is a* colouring *of $N(v)$ satisfying:*

1. *A gray colour is assigned to each neighbour $V_H(w)$ of $v \in PDS$ that has a* forward edge *incident from v to $w \in N(u)$ where $u \in PDS$ (see Subfig 5.2.(a)):*
$$\exists v, u \in PDS : \Big(\exists w \in N(u) : E_f(vw)\Big)$$
2. *An orange colour is assigned to each neighbour $V_H(w)$ of $v \in PDS$ with a* cross edge *satisfying:*
   a. *There exists a* cross edge (vw) *that is incident from $v \in PDS$ to $w \in N(u)$ where $u \in PDS$ (see Subfig 5.2.(b)):*
   $$\exists (vw) \in E_c : \Big((v, u \in PDS) \wedge (w \in N(u))\Big)$$
   b. *There exists a* cross edge (uw) *between the subtrees of an ancestor $v \in PDS$ incident from $u \in N(v)$ to $w \in N(v)$ and the vertex u is a leaf or has no out-red edge (see Subfig 5.2.(c)):*
   $$\exists (uw) \in E_c : \Big((v \in PDS) \wedge (u, w \in N(v)) \wedge (d^+_E(u) = 0 \vee d^+_{E^r}(u) = 0)\Big)$$
   c. *There exists a* cross edge (vw) *incident from a leaf vertex v such that $E^+_t(v) = 0$ to $w \in N(u)$ where $u \in PDS$ (see Subfig 5.2.(d)):*
   $$\exists (vw) \in E_c : \Big((u \in PDS) \wedge (w \in N(u)) \wedge (E^+_t(v) = 0)\Big)$$
   d. *There exists a* cross edge (vw) *incident from a vertex v with no out-red edge such that $d^+_{H^r}(v) = 0$ to a vertex $w \in N(u)$ where $u \in PDS$ (see Subfig 5.2.(e)):*
   $$\exists (vw) \in E_c : \Big((u \in PDS) \wedge (d^+_{H^r}(v) = 0) \wedge (w \in N(u))\Big)$$

We observe that we have at most $2^{N(v)}$ colouring states for the neighbours of the $v \in PDS$. We therefore seek to employ a *Valid Colouring* of $N(v)$ to reduce the state space by utilising *forward* and *cross edges* drawing on the following lemma due to Guo [49].

Lemma 5.2 (A Valid Orientation of an Undirected Graph, Guo *et al.* [49])
Let $C = (V, E^r)$ be a Valid Colouring *of a directed graph G with origin $\Phi \subseteq V$:*

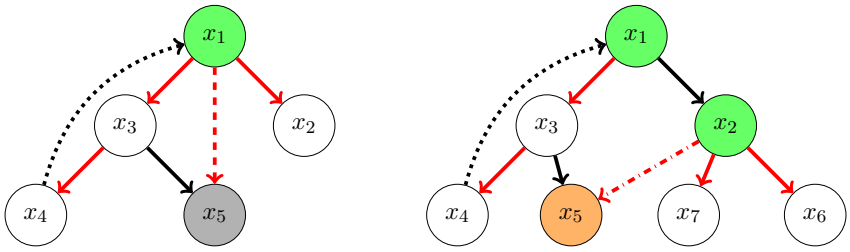
1. *For each $v \in V \setminus \Phi$, there is exactly one directed path from the vertices in Φ to v.*
2. *Two directed paths from Φ to distinct vertices in $V \setminus \Phi$ are vertex-disjoint with the possible exception of their tail endpoints in Φ.*

Together with Lemma 5.2, we prove the following lemma:
Lemma 5.3 (The Number of Dependency Paths in a *Valid Colouring*)
Given a PDS for H where $S \subseteq V$ in $H(V)$, the number of dependency paths *of $v \in S$ is equal to the number of neighbours of v.*

Proof. Given a PDS for $H = (V, E)$, satisfying the same assumptions (1,2 and 3) as G' shown in Subsection 1.4.2 in Chapter 1, where $S \subseteq V$ in $H(V)$. Supposing that $v \in S$, by applying the first Directed rule **D1**, defined in Definition 2.16, to a PDS, all neighbours of v are power dominated such that for each vertex $u \in N(v)$, there is exactly one directed edge from the vertex v to u. For instance, if $v \in S$ has the out-degree of 3 (i.e. $d^+(v) = 3$), then the number of *dependency paths* that are incident from v is also 3.
Assuming that there exists $w \in N(v)$ that is not covered yet. In this case, w should be power dominated in two ways:

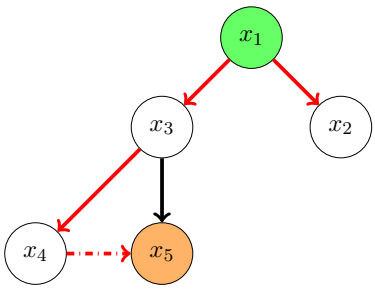
a. There exists en edge (zw) incident from $z \in S$ to w. Since there is no another edge incident to w except the one coming from v, w is still not controlled, or
b. there exists $x \in V \setminus S$, where x is already controlled, that has an edge $(e = xw)$ to w, and that would imply $d^+(x) > 1$ which is not allowed by Definition 5.6.

Thus, all the neighbours of v should be controlled by v, meaning the number of directed paths that are incident from v is the same as the number of the neighbours of v. As a result, the minimisation of the covered neighbours by v results in the reduction of *dependency paths*, and therefore it leads to reduce a PDS in H.

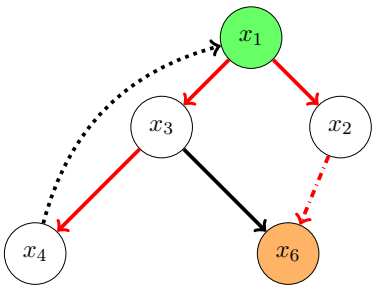


(a) Forward edge

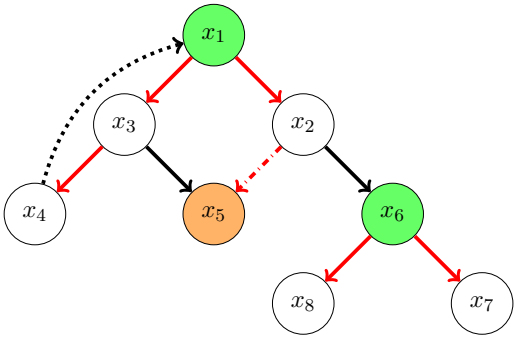
(b) Cross edge (1)



(c) Cross edge (2)



(d) Cross edge (3)



(e) Cross edge (4)

Figure 5.2: Case Enumeration for Colouring of the Neighbours of a PDS in a DFS

Definition 5.9 (Minimising a PDS by Colouring Forward and Cross Edges in a DFS) *Edges* (uw) *are coloured red if and only if:*

1. *It is a* forward edge (uw) *incident from* $u \in PDS$ *to* $w \in N(v)$ *where* $v \in PDS$ *(see Subfig 5.2.(a)) such that* $\exists e = (uw) \in E_f : \Big((u, v \in PDS) \wedge (w \in N(v))\Big)$
2. *It is a* cross edge (uw) *incident from* $u \in PDS$ *to* $w \in N(v)$ *where* $v \in PDS$ *(see Subfig 5.2.(b)) such that* $\exists e = (uw) \in E_c : \Big((u, v \in PDS) \wedge (w \in N(v))\Big)$
3. *It is a* cross edge (uw) *between the subtrees of an ancestor* $v \in PDS$ *incident from* $u \in N(v)$ *to* $w \in N(v)$*, and the vertex* u *is a leaf or has no out-red edge (see Subfig 5.2.(e)):*

$$\exists(uw) \in E_c : \Big((v \in PDS) \wedge (u, w \in N(v)) \wedge (d^+_E(u) = 0 \vee d^+_{E^r}(u) = 0)\Big)$$

4. *It is a* cross edge (uw) *incident from a leaf vertex* ($E^+_t(u) = 0$) *to* $w \in N(v)$, *where* $v \in PDS$ *(see Subfig 5.2.(d)):*

$$\exists e = (uw) \in E_c : \Big((E^+_t(u) = 0) \wedge (v \in PDS) \wedge (w \in N(v))\Big)$$

5. *There exists a* cross edge (uw) *incident from a vertex* u *with no out-red edge such that* $d^+_{H^r}(u) = 0$ *to a vertex* $w \in N(v)$ *where* $v \in PDS$ *(see Subfig 5.2.(e)):*

$$\exists(uw) \in E_c : \Big((v \in PDS) \wedge (d^+_{H^r}(u) = 0) \wedge (w \in N(v))\Big)$$

Because of insufficient *forward/cross edges* that are incident to the neighbours of a PDS, it is necessary to identify a criterion of addition red edges from a PDS to its neighbours as stated in the following lemma:

Lemma 5.4 (The Addition of Red Edges to the Neighbours of a PDS in a DFS) *Given a DFS and* Valid Colouring *for* $H = (V, E)$. *If there is a lack of* forward/cross edges *that are incident to the neighbours of a PDS, then one can add a red edge to obtain a minimal PDS in* H *satisfying:*

1. *A red* forward edge *is assigned from* $v \in PDS$ *to each neighbour* $V_H(w)$ *of* $u \in PDS$ *such that there is a red edge directed from* v *to* $w \in N(u)$ *(see Subfig 5.3.(a)).*
2. *A red* cross edge *is assigned from* $v \in PDS$ *to each neighbour* $V_H(w)$ *of* $u \in PDS$ *where a* cross edge *satisfies:*
   a. *A red* cross edge (vw) *is assigned from* $v \in PDS$ *to* $w \in N(u)$ *where* $u \in PDS$ *(see Subfig 5.3.(b)).*
   b. *A red* cross edge (uw) *is assigned between the subtrees of an ancestor* $v \in PDS$ *that is incident from* $u \in N(v)$ *to* $w \in N(v)$ *and the vertex* u *is a leaf or has no out-red edge that is incident to the neighbours of a PDS (see Subfig 5.3.(c)).*
   c. *A red* cross edge (vw) *is assigned from a leaf vertex* v *such that* $E^+_t(v) = 0$ *to* $w \in N(u)$ *where* $u \in PDS$ *(see Subfig 5.3.(d)).*
   d. *A red* cross edge (vw) *is assigned from a vertex* v *with no out-red edge such that* $d^+_{H^r}(v) = 0$ *to a vertex* $w \in N(u)$ *where* $u \in PDS$ *(see Subfig 5.3.(e)).*

Proof. We are assume that a DFS for $H = (V, E)$, satisfying the same assumptions (1,2 and 3) as G' mentioned Subsection 1.4.2 in Chapter 1, and a *Valid Colouring*

of a *Directed PDS* for H are given. We prove that the above constraints in this lemma, applied to the neighbours of a PDS through adding *forward/cross edges* to its neighbours, result in the minimisation of a PDS.

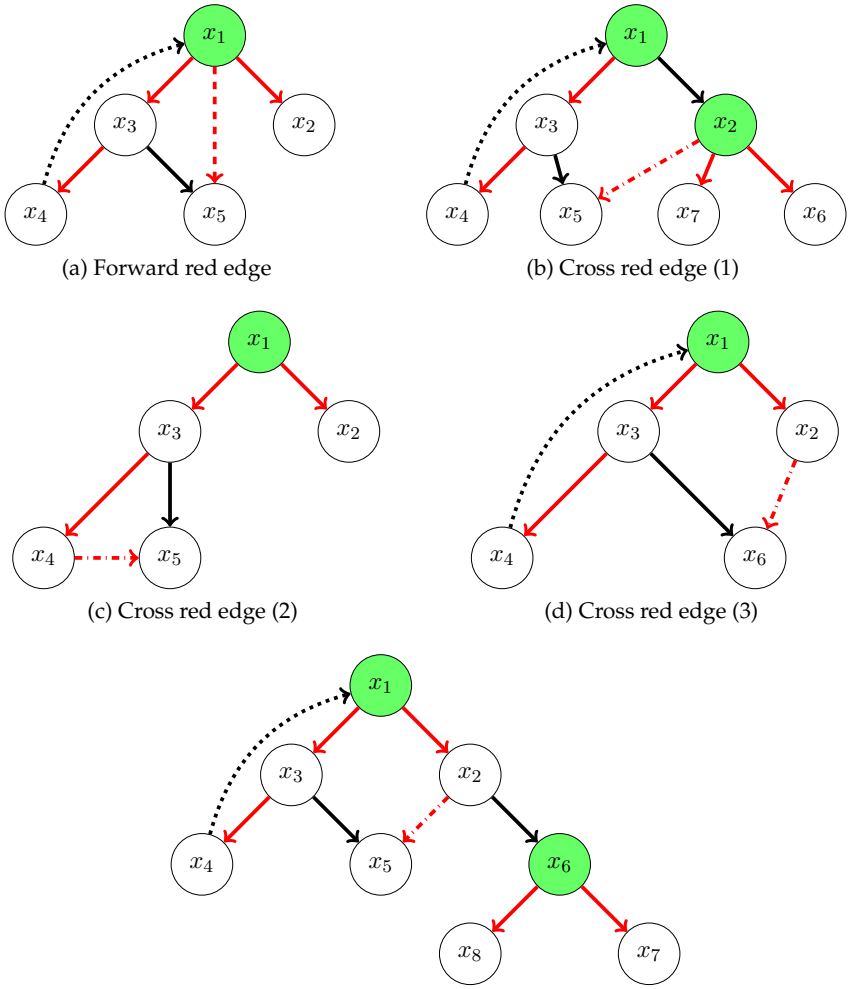
Let a vertex $x_j \in PDS$ is a subtree of the root of a DFS, denoted by $x_i \in PDS$. We claim that a *forward edge* (E_f) should be directed from x_i to the neighbours of a vertex x_j (i.e. $x_i \rightarrow w \in N(x_j)$) in order to minimise the number of a PDS as stated in the first constraint of this lemma. Let E_f is not assigned from x_i to w. Instead, E_f is directed from x_i to any vertices except those in $N(x_j)$. Therefore, the neighbours of x_j are still power dominated by a vertex x_j. Hence, there is no change in the number of a PDS and E_f should be assigned from x_i to vertices in $N(x_j)$.

Now let x_j and x_k are the subtrees of x_i and $\{x_i, x_j, x_k\} \in PDS$. We claim that a *cross edge* (E_c) should be assigned from a vertex x_j to each neighbour of x_k in order to minimise the number of a PDS as stated in the second constraint of this lemma. Now we apply the same argument above to a *cross edge* (E_c) and suppose that E_c is directed from x_j to the vertices in the subtree x_k except its neighbours (i.e. $x_j \rightarrow w \notin N(x_k)$). Thus, the neighbours of x_k are still controlled by a vertex x_k. Hence, there is no difference in the number of a PDS and E_c should be directed from x_j to vertices in $N(x_k)$ to obtain a minimal PDS.

Let assume there is E_c between the neighbours of the subtree x_j as shown in Subfig (5.3.c). Therefore, if E_c is coloured by red and there exists another *cross edge* incident to the neighbours of x_j is also coloured by red, then the vertices that are downstream of $N(x_j)$ will no be longer controlled (i.e. $V \setminus N[x_j]$). Hence, E_c should be directed to the neighbour of x_j that is a leaf vertex or has no out-red edge that is incident to the neighbours of x_j as stated in the third constraint of this lemma.

Now let consider Subfig (5.3.d), and suppose that E_c is incident from a vertex $w \in x_i$, where w is not a leaf to a vertex $z \in x_j$ (i.e. it has an out-red edge incident to anther vertex). By colouring E_c to red, a vertex w is already power dominated another vertex meaning w should be in a PDS; hence E_c should be assigned from a leaf vertex in order to avoid the increase of the number of a PDS as stated in the

forth constraint of this lemma. However, if w has an out-edge that is incident to a vertex in a PDS, then E_c can be coloured and directed from w to z as stated in the last constraint of this lemma and shown in Subfig (5.3.e).



(a) Forward red edge

(b) Cross red edge (1)

(c) Cross red edge (2)

(d) Cross red edge (3)

(e) Cross red edge (4)

Figure 5.3: Case Enumeration for Adding Red Edges to the Neighbours of a PDS in a DFS

We can now formulate the relationship between a DFS tree and a PDS:

Theorem 5.5 (A PDS in a DFS Structure)

Given a DFS for $H = (V, E)$, a colouring *of the neighbours of $v \in PDS$ depending on the* forward *and* cross edges *will yield improved average-case complexity of a PDS in H.*

Proof. Let $H = (V, E)$ be a directed graph satisfying the same assumptions (1,2 and 3) as $G^{'}$ shown in Subsection 1.4.2 in Chapter 1. We are given a DFS structure for $H = (V, E)$ and its a *Valid Colouring* matching Subfig 5.2.(a).

If we apply the rules of a PDS to H, regardless of *forward* and *cross edges*, then $\{x_1, x_3\}$ are a PDS in H. Since there is a *forward edge* that is incident from $x_1 \in PDS$ to a neighbour x_5 of x_3, then one can take an advantage of the edge to minimise the number of a PDS to become one. Thus, x_5 should be coloured to achieve a minimum PDS in H.

Now, supposing there exists a *cross edge* in H. According to Definition 5.9, there are four cases of *colouring* a *cross edge*. Consider only the first case matching Subfig 5.2.(b), by applying the same argument above, the number of a PDS in H can be $\{x_1, x_2, x_3\}$. Because of the *cross edge* (x_2x_5), the vertices in a PDS are reduced to $\{x_1, x_2\}$, hence in this case, a vertex x_5 that a *cross edge* is incident to it should be coloured. We observe that a directed graph has a cycle if and only if there is a *back edge*; as a result, it can form a *dependency cycle* in H. Thus, a *back edge* is not considered in a *Valid Colouring* of H (see Subfig 5.2.(a), where *colouring* the *back edge* (x_4x_1) leads to a *dependency cycle* in H). Hence, a *Valid Colouring* of $H = (V, E)$ has no *dependency cycle* (see Definition 5.6).

We now formulate a bottom-up dynamic programming algorithm generating a PDS for $H = (V, E)$ based on earlier results [2, 49, 13] and the following theorem:

Theorem 5.6 (Partition Colouring in a DFS)
Given a directed graph $H = (V, E)$ constructed by a DFS and $S \subseteq V(H)$, S power dominates H if and only if there is a Valid Colouring *of H with S as the set of origins.*

Proof. Given a DFS structure for a directed graph $H = (V, E)$, satisfying the same assumptions (1,2 and 3) as $G^{'}$ shown in Subsection 1.4.2 in Chapter 1. Assuming that $S \in V(H)$ is a *Directed PDS* of $H = (V, E)$; thus, $P(S) = V(H)$. Let apply a *Valid Colouring* C with S as the set of sources by *colouring* the edges in H according to the degree constraints in Definition 5.6. We colour an edge (uv) red from

u toward v if either u is a domination vertex and v is controlled by applying the first rule **D1** to u, or vertex v is controlled by applying the second rule **D2** to u, as defined in Definition 2.16. Note all possible domination and the propagation rules to S should be in order. Moreover, we do not apply **D1** or **D2** to control previously covered vertices. It is easy to check that with this *colouring* the degree constraints of Definition 5.6 are satisfied.

We can now show by contradiction that there is no *dependency cycle* in a *Valid Colouring*. Let $u \to v$ denote a vertex v power dominated by a vertex u, supposing further that $C = u_1, u_2, \ldots, u_m$ is a *dependency cycle* and all red edges in C are in the same direction. Red edges (u_i, u_{i+1}) imply that $u_i \to u_{i+1}$ for all $i = 1, 2, \ldots, m-1$; then we obtain $u_1 \to u_2 \to \cdots \to u_m$, but this is a contradiction since the last red edge from u_m back to u_1 implies that $u_m \to u_1$. Note that *back edges* are not considered in a *Valid Colouring* of a DFS; therefore, *back edges* are not coloured red. Hence, there is no *dependency cycle* with all edges coloured red. The final result is that H has a *Valid Colouring* $C = (V, E^r)$ with $S \subseteq V(H)$ as the set of origins.

Together with Theorem 5.6, we immediately obtain our main result as follows:

5.5 Time Complexity

Lemma 5.7 (Time Complexity of Re-using a Remaining PDS Structure)
Given a Valid Colouring *of a DFS for* $H = (V, E)$*, one can improve an average-case complexity of computing a PDS in* $\mathcal{O}(|V + \Big\{E_t \cup E_f\big[N(v)\big] \cup E_c\big[N(v)\big]\Big\} \setminus E_b|)$ *time, where* $v \in PDS$.

Proof. We are given a *Valid Colouring* of a DFS for a directed graph $H = (V, E)$, satisfying the same assumptions (1,2 and 3) as $G^{'}$ shown in Subsection 1.4.2 in Chapter 1. The time complexity of the algorithm, satisfying the assumption (4) as shown in Subsection 1.4.2 in Chapter 1, is to compute a PDS for H by re-using remaining fragments of the original graph where possible and identifying previously un-used edges to minimise the number of a PDS.

For the best case, the running time is to traverse only *tree edges* in a given DFS, where there are at most $|V + \{E_t\}|$ edges. In the average case, as *back edges* E_b are not computed while applying a *Valid Colouring* to a given DFS structure. Therefore, the most running time of this algorithm is to consider only all possible cases for *colouring* the neighbours of a PDS with existence *forward/cross edges* which are incident to the neighbours of a PDS, as set forth in Definition 5.8. Consequently, the number of *forward/cross edges* that should be traversed is at most $|V + \{(E_f[N(v)] \cup E_c[N(v)\}]) \setminus E_b|$ edges.

We give this result also in constructive form in the following algorithm:

Algorithm 5.1: Generation of a Minimum PDS via a DFS Structure

Input : Given a DFS for $H = (V, E)$, with a DFS traversal resulting in tree T based on tree edges
Output: A minimum PDS of H

Let the inner vertices of T be sorted as L based on tree edges in post-order traversal of T, where r is a root of T and bottom-up DP from leaves to root;
Let $S \leftarrow \emptyset$;
while $L \neq r$ **do**
    $v \leftarrow$ the first vertex in L; $L \leftarrow L \setminus \{v\}$;
    if $d^+(v) \geq 2$ *has uncovered children* **then**
        if (v) *is a cut-vertex* **then**
            $S \leftarrow S \cup \{v_{ct}\}$;
            Apply a *Valid Colouring* for all vertices that are reachable from v;
        else
            $S \leftarrow S \cup \{v\}$;
            Apply a *Valid Colouring* for $H = (V, E)$ in a DFS;
    forall the $v \in PDS$ **do**
        if $w \in N(v)$ **then**
            Minimising S by colouring w and its forward and cross edges;
if r *is uncovered* **then**
    while r *has* $e \in E_f$ *or* $e \in E_c$ *that is incident to* $N(S)$ **do**
        Applying a *Valid Colouring* for $H = (V, E)$ in a DFS;
        Minimising S;
    $S \leftarrow S \cup \{r\}$;
return S;

5.6 Summary

The timely recovery of control as represented by *structural controllability* (for LTI systems) after a control graph has been damaged such as following an attack is a significant problem in control systems. If control can be recovered entirely or to the largest extent possible, the potential service degradation or damage caused by an attacker can be reduced substantially, or attackers can be kept from taking over a network and control over it entirely. This, however, requires the ability to recover *controllability* as fast as possible since adversaries may — particularly where such attacks occur after a substantial period of intelligence-gathering — repeatedly attack even while recovery operations are still in progress.

In this chapter we therefore proposed a novel algorithm based on a DFS structure, which yields an improved average-case complexity over previous work in Chapter 4, after an event or attack leading to a degradation of the control of the network and a significant reduction of its observability. This DFS-based approach reduces the average-case complexity of the recovery algorithm by re-using remaining fragments of the original, efficient control graph where permitted whilst identifying previously un-used edges to minimise the number of a PDS.

Chapter 6

Recovering Structural Controllability in the Presence of Compromised Nodes

6.1 Overview

Large-scale distributed control systems such as those encountered in electric power networks or industrial control systems could be vulnerable to attacks, in which adversaries can compromise controllability of dependent nodes, and therefore, disconnect parts the of control graph.

In this chapter we address the question of how to recover a control graph as far as possible in the presence of such *compromised nodes*. The proposed approach is based on a BLOCK DECOMPOSITION of a directed graph, allowing us to identify *Cut-Vertices* (or *articulation points*) and cut-edges, where *structural controllability* for a given $G^{'}$ can be restored in the presence of *compromised nodes* in $\mathcal{O}(nc^{W})$ time for a constant c, where W denotes *dependency paths* that are the remainder of a *compromised node* (i.e. dependent nodes). This algorithm results in the recovery of a PDS structure, and ultimately the re-gaining of control for operators of control systems by applying three phases.

Definition 6.1 (A Compromised Node)

A vertex $v \notin PDS$ in a Valid Colouring *is said to be a* compromised node, *if there is **no** red in-edge incident to it such that $d^{-}_{G^{'}_{r}}(v) = 0$, where $G^{'}_{r}$ denotes a directed graph with only red in/out-edges. Otherwise, a vertex $v \in PDS$ in a* Valid Colouring *is a* compromised node *where there is **no** red in/out-edge incident to v such that $d^{-}_{G^{'}_{r}}(v) = 0$ and $d^{+}_{G^{'}_{r}}(v) = 0$.*

6.2 Problem Statement and Assumption

In real-world context, the logical structures of PDS-based networks and real-world monitoring systems have a remarkable similarity, where driver nodes (as represented by a PDS in in the context of electrical power network control) can represent *e.g.* control terminal units that control industrial sensors or actuators. This chapter only considers a directed graph $G^{'} = (V, E)$, generated by $ER(n, p)$, where $G^{'} = (V, E)$ represents *e.g.* an electrical power system including a set of nodes (buses) and a set of edges (transmission lines) connecting the buses. A bus is a substation where transmission lines are joined. A power system also includes a set of generators, supplying power and a set of loads, into which the power is directed.

We assume that $G^{'}$ satisfies the assumptions (1,2 and 3) as shown in Subsection 1.4.2 in Chapter 1. Let $S^{'}$ be a given *Directed PDS* of $G^{'}$ that is constructed by applying a *Valid Colouring* approach in terms of colouring edges to blue and red. Assume an adversary with sufficient knowledge of the network distribution is able to compromise a subset of nodes in $G^{'}$ (e.g. $v \in PDS$ or $v \notin PDS$). In real scenarios, suppose the actuators that have the ability to control more states in a power network are the most valuable targets for the attacker, where the failure of these components may have serious consequences for collecting data from certain sensors or the information flow of the actuators); in the context of the PDS problem, we call the failures of the most valuable nodes or its dependents (*e.g.* remote terminal units that control industrial sensors or actuators) in this chapter *compromised nodes*. Therefore, a PDS-based network should be repaired more efficiently by recomputing a PDS for $G^{'}$ when the leftover nodes are not power dominated after the detection of *compromised nodes* without re-applying a *Valid Colouring* to the whole $G^{'}$ from the beginning.

•Input:

Given a *Valid Colouring* for a *Directed PDS* in $G^{'}$ and *compromised nodes* such as $v \in PDS$ or $v \notin PDS$

•Question:

Can we recover a control graph as far as possible in the presence of such compromised nodes if the PDS or its dependent nodes have been partially compromised without re-applying a *Valid Colouring* to $G^{'}$ from the beginning?. A real world scenario related to this problem is how to restore overall controllability of an electric power system if remote terminal units that control industrial sensors or actuators have been partially violated without complete re-computation.

•Output:

Recovering the leftover vertices of *compromised nodes* in case of $v \in PDS$ or $v \notin PDS$ after an attack on a control graph through:

i. Decomposing a directed graph $G^{'}$ into k-separable set of *Blocks* based on blue and red edges (see Section 6.3):
   a. Finding *Edge-Cut Set* via considering blue edges.
   b. Identifying *Cut-Vertices* (or *articulation points*) giving an equivalent formulation for a PDS in directed graphs as in [14] via considering red edges.

ii. Re-using blue edges existing in a *compromised Block*, allowing us to recover *structural controllability* of a control graph (see Subsection 6.4.1), or

iii. using blue edges that are incident to the *compromised Block*, allowing us to reconstruct the control graph as far as possible (see Subsection 6.4.2), or

iv. identifying criteria for the efficient addition of red edges into a *compromised Block*, (see Subsection 6.4.3).

6.3 Reconstructing DPDS via Block Decomposition

A BLOCK DECOMPOSITION of a graph, denoted by $B(G^{'})$, can be used to give equivalent characterisations of trees through identifying *Cut-Vertices* (or *articulation points*) which its removal (together with the removal of any incident edges) result in a disconnected graph, and identifying *Blocks* of a directed graph.

Definition 6.2 (A Block)

A Block *is a maximal connected subgraph with no* Cut-Vertex.

Definition 6.3 (A Cut-Vertex or an Articulation Point in a Block)

Let $G^{'}$ be a graph with $k(G^{'})$ components. A vertex v of $G^{'}$ is called a Cut-Vertex *or an* articulation point *(**C**) of $G^{'}$ if $k(G^{'}-v) > k(G^{'})$.*

Definition 6.4 (A Weakly Red-Connected Component in a *Valid Colouring*)

Red-Connected Component, *denoted by X, is defined as exactly one directed Path (P) weakly connected by red edges (e^r) with no directed cycle $X = \{v_1, e^r_1, v_2, e^r_2, \ldots, e^r_{i-1}, v_i\} = P$ such that for every pair vertices of X, there is an undirected path from v_i to v_1 such that all the edges in P are directed from v_1 to v_i and coloured by red.*

Definition 6.5 (A Set of Weakly Red-Connected Components)

In a Block, *a set of* Red-Connected Components, *denoted by S, is defined as a set of directed paths connected by red edges with no directed cycle, such that $S = \{X_1, X_2, \ldots, X_i\}$. We call a set of S that shares the same* tail *endpoint $v \in PDS$ as a* Block, *where each* Block *has at least one* Red-Connected Component.

Definition 6.6 (A Leaf and Tail of a Weakly Red Connected Component)

A leaf vertex *(i.e. a* head vertex*) is the vertex with no red out-edge, whereas the* tail vertex *is the vertex with a red in-edge that is incident from $v \in PDS$ to the* tail vertex.

Example 4 Consider Figure 6.1, a *Block* with $x_{13} \in PDS$ has two RCCs starting from x_{13}, $P_1 = \{x_{14}, x_{17}\}$ and $P_2 = \{x_{15}, x_{16}\}$, where (x_{14},x_{15}) are the tails of P_1 and P_2 respectively and (x_{16},x_{17}) are the leaves (heads) of P_1 and P_2 respectively. □

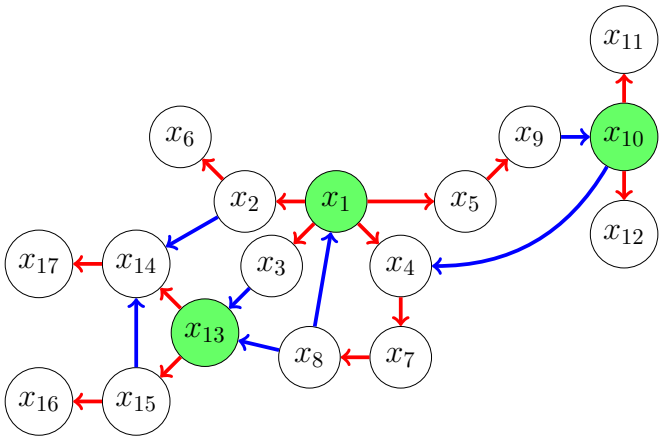
Definition 6.7 (A Blue Edge-Cut Set in a *Valid Colouring*)

Given a Valid Colouring *for a* Directed PDS *in $G^{'}$. A* Blue Edge-Cut Set, *denoted by Y, of $G^{'}$ is a set of blue edges satisfying (see Subfig 6.3.(a)):*

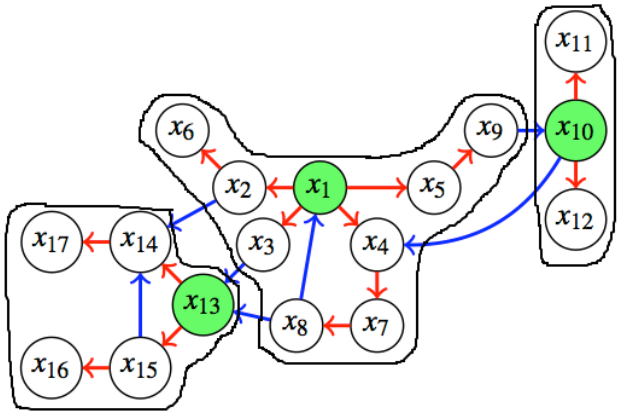
1. *The removal of all blue edges in Y disconnects $G^{'}$ to k-separable set of* Blocks *such that each* Block *is connected by red edges.*
2. *The removal of some (but not all) of blue edges in Y does not disconnect $G^{'}$.*

Note that there may some blue edges that are not belonging to a *Blue Edge-Cut Set* since they are not satisfying the constraints in Y such as the edge ($x_{15}x_{14}$) and (x_8x_1)

in Figure 6.1. Therefore, they do not consider when computing a *Blue Edge-Cut Set* in a *Valid Colouring*.



(a) The valid colouring of a directed graph



(b) Blocks of a directed graph

Figure 6.1: Re-Construction of Blocks of a Directed graph via Red Edges

Definition 6.8 (A Cut-Vertex (or an Articulation Point) in a *Valid Colouring*)
Given a Valid Colouring *for a* Directed PDS *in* $G^{'}$. *A vertex* $v \in PDS$ *of* $G^{'}$ *is called a* Cut-Vertex *(or an* articulation point*) (**C**) of* $G^{'}$ *if the removal of* v *with addition of a* Blue Edge-Cut Set Y *disconnects* $G^{'}$ *to k-separable set of* Blocks *(see Subfig 6.3.(a)).*

Definition 6.9 (A Block of a Directed PDS in a *Valid Colouring*)
A Block-Vertex *(**B**) of a directed graph is a maximal connected subgraph by red edges such that a set of* Red-Connected Components *with no* Y*, that is reachable from a vertex in a PDS, such that a sequence of vertices connected by red edges forming a* dependency path *starts from* $v_1 \in PDS$ *where* $B = \{v_1, e^r_1, v_2, e^r_2, \ldots, e^r_{i-1}, v_i\}$.

Definition 6.10 (A Block Cut-Vertex Tree (T^B) of a Directed PDS)

Suppose a Valid Colouring *for a* Directed PDS *in* $G^{'}$ *is given. Let* B_k *be the set of* Blocks *and* C_k *be the set of* Cut-Vertices *of* $G^{'}$*. Constructing a graph* T^B *with vertex set* $C_k \cup B_k$ *as an ordinary tree regardless of the fact that every* Block-Vertex *is actually subset of vertices of the original graph such that* $c_i \in C$ *is adjacent to* $b_j \in B$ *if and only if the* Block b_j *of* $G^{'}$ *contains the* Cut-Vertex c_i *of* $G^{'}$*.*

Lemma 6.1 (A Construction of a Block Cut-Vertex Tree)

Given a Valid Colouring *for a* Directed PDS *in* $G^{'}$*, reconstructed in terms of red and blue edges, one can construct a* Block Cut-Vertex Tree *(*T^B*) of a* Directed PDS *by identifying* Cut-Vertices *and* Blocks *of a* Directed PDS*.*

Proof. Given a *Valid Colouring* for a *Directed PDS* in $G^{'}$, satisfying the assumptions (1,2 and 3) as shown in Subsection 1.4.2 in Chapter 1. According to Definition 6.10, $T^B(G^{'}) = C_k \cup B_k$. We show how to obtain (B_k). A *Block* (B) of a *Directed PDS* is identified by applying Definition 6.7, where the removal of a *Blue Edge-Cut Set* will disconnect $G^{'}$ into a set of *Red-Connected Components* (S), that are reachable from $v \in PDS$, as in Definition 6.5 (i.e. a maximal connected subgraph by red edges) (see Subfig 6.3.(a)). Therefore, each S is as *Block*.

Now we define C_k; according to the constraints in Definition 4.6, each vertex in a *Valid Colouring* has at most one red in-edge except a vertex $u \in PDS$ that may have at least one blue in-edge $e_b(wu) \in Y$ such that the removal $e_b(wu)$ will disconnect $G^{'}$ (see green vertices in Subfig 6.3.(a)), hence, u is a *Cut-Vertex*. By way of contradiction, assuming that not all of a *Blue Edge-Cut Set* are omitted (see Subfig 6.3.(b)). Thus, there may exist some *Blocks* that are still connected to $G^{'}$ via blue edges in S which contradicts Definitions 6.7 and 6.9.

The proof of the following lemma is similar to the proof of Lemma 6.1.

Lemma 6.2 (A Construction of a Block Cut-Vertex Tree of a Directed PDS)

Given a Valid Colouring *for a* Directed PDS *in* $G^{'}$*. When drawing* Block Cut-Vertex Tree *(*T^B*) of a* Directed PDS*, we represent (see Figure 6.2):*

1. Cut-Vertex*, which has an equivalent formulation for a PDS by green colour.*

2. *A tree red/blue edge, which describes a relation between a vertex and one of its direct descendants, by a solid line* _____ *satisfying:*
   i. *Each* Cut-Vertex *has exactly one solid red out-edge.*
   ii. *Each* Cut-Vertex *may have at most one solid blue in-edge or may have no solid blue in-edge.*
   iii. *Each* Block *may have at least one solid blue out-edge which is incident to* C_k*, or have no solid blue out-edge.*
   iv. *Each* Block *has exactly one solid red in-edge.*
3. *The remaining blue in/out edges are represented by a dashed line* - - - - -

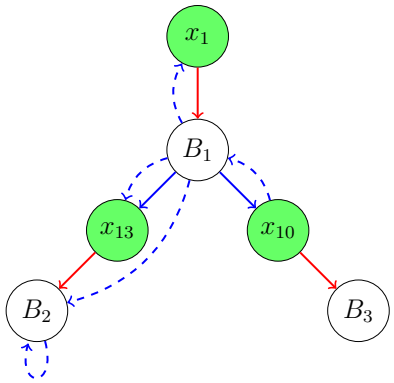


Figure 6.2: A Construction of a *Block Cut-Vertex Tree* (T^B) of a *Directed PDS*

Example 5 Consider a graph in Figure 6.1 and how is reconstructed as *Block Cut-Vertex Tree* as shown in Figure 6.2. The *Blocks* are

- $B_1 = \{x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9\}$.
- $B_2 = \{x_{14}, x_{15}, x_{16}, x_{17}\}$.
- $B_3 = \{x_{11}, x_{12}\}$.

Furthermore, *Cut-Vertices* which have the equivalent formulation for a PDS are

- $C_1 = \{x_1\}$.
- $C_2 = \{x_{13}\}$.
- $C_3 = \{x_{10}\}$.

According to Lemma 6.2, there is only one solid red out-edge that is incident from $C_k \rightarrow B_k$ such as $\{(x_1 \rightarrow B_1), (x_{13} \rightarrow B_2), (x_{10} \rightarrow B_3)\}$, or may have at most one solid blue in-edge such as $\{x_{10}, x_{13}\}$ or no solid blue in-edge such as $\{x_1\}$.

Moreover, there may have at least one solid blue out-edge that is incident from $B_k \to C_k$ such as $\{(B_1 \to x_{13}), (B_1 \to x_{10})\}$, or have no solid blue out-edge such as $\{B_2, B_3\}$. Note that blue in/out edges have no effect on the *structural controllability* of $G^{'}$ as we only consider red edges for obtaining full control of $G^{'}$. Thus, blue in/out edges are still existing in $G^{'}$ as a part of connectivity. This, however, helps to use blue in/out edges (represented by a dashed line) to recover the *structural controllability* of a graph. □

We formulate the relationship between T^B and a PDS together with Lemma 6.1:

Theorem 6.3 (A PDS in a Block Cut-Vertex Tree)

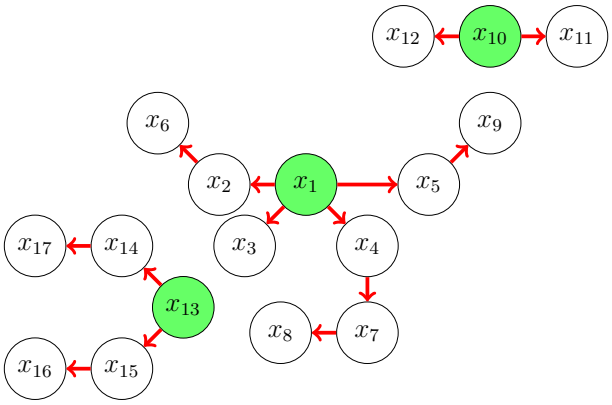
Given a Valid Colouring *for a* Directed PDS *in* $G^{'}$*, reconstructed as a* Block Cut-Vertex Tree *(*T^B*), a* Cut-Vertex *(*C*) is said to be in a PDS if and only if:*

1. *The removal of all blue edges in* Y *disconnects* $G^{'}$ *to k-separable set of* Blocks *connecting by red edges.*
2. *There is no* dependency cycle *in each* Block *such that there is no a red edge* $(v_{i+1}, v_i) \in B_k$ *incident from any* head *endpoint* v_{i+1} *to the vertex* $v_i \in PDS$ *in the same* Block.
3. *A vertex* $v_1 \in PDS$ *has at least one RCC i.e.* $\{v_1, e_1^r, v_2, e_2^r, \ldots, e_{i-1}^r, v_i\} = P$.

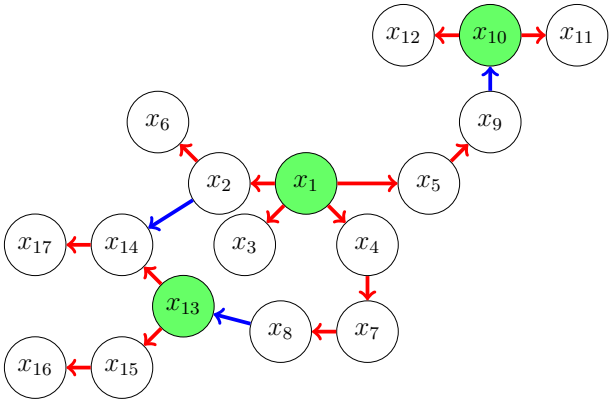
Proof. We are given a *Valid Colouring* for a *Directed PDS* in $G^{'}$ reconstructed as a *Block Cut-Vertex Tree*, satisfying the assumptions (1,2 and 3) as shown in Subsection 1.4.2 in Chapter 1. We prove the theorem by introducing three cases as follows:

1. **Condition (1):** Consider Figure 6.3, we prove that a PDS will be changed in case of not all of Y are omitted. By applying the propagation rule as defined in Definition 2.16; the blue edge (x_8, x_{13}) should be coloured to a red edge in order to control $\{x_{13}, x_{15}, x_{16}\}$ in sequence. However, x_{13} will no longer be in a PDS; thus, x_{14} should be controlled by x_2 that will be in a PDS in order to control $\{x_6, x_{14}\}$ simultaneously. Hence, a PDS has changed to become $\{x_1, x_{10}, x_2\}$.
2. **Condition (2):** We show by way of contradiction that there is no *dependency cycle* in T^B. Let $u \to v$ denotes a vertex v is covered after u; assuming that $X = u_1, u_2, \ldots, u_m$ is a *dependency cycle* such that all red edges in X are

in the same direction and $u_1 \in PDS$ has a red in-edge which is incident from u_m. Therefore, the red edge (u_i, u_{i+1}) implies that $u_i \rightarrow u_{i+1}$ for all $i = 1, 2, \ldots, m-1$; thus we get $u_1 \rightarrow u_2 \rightarrow \cdots \rightarrow u_m$, but this contradicts with the covering rules as the *head* u_m has a red edge that is incident to u_1 implying $u_m \rightarrow u_1$. Consequently, we prove Condition (3), where vertices in a PDS must have no red in-edge according to Definition 4.6. As proved that there is no *dependency cycle* in T^B (as shown in Condition 2), then all vertices in X are reachable from $u_1 \in PDS$; thus, X is power dominated by u_1.



(a) Removal of Blue Edge-Cut set



(b) Removal of **some** Blue Edge-Cut set

Figure 6.3: Case Enumeration for the Removal of a *Blue Edge-Cut Set*

6.4 The Process of Recovering Structural Controllability

The algorithm is divided into three phases for recovering *structural controllability* in the presence of *compromised nodes*. We consider (in order of priority) the blue edges that are inside a *compromised Block* itself, or incident to a *compromised Block*.

We recall each *Block* has at least one RCC, where $X = \{v_1, e_1^r, \ldots, e_{i-1}^r, v_i\}$ or a set of RCCs, where $S = \{X_1, X_2, \ldots, X_i\}$. The *tail* and *head vertices* of X are denoted by $t(v)$ and $h(v)$, respectively. A sequence of vertices that are reachable from a *compromised node* in X is denoted by W, where a *compromised node* may be $v \in PDS$ or $v \notin PDS$; thus, there are two cases of recovering *structural controllability* via internal blue edges of a *Block*.

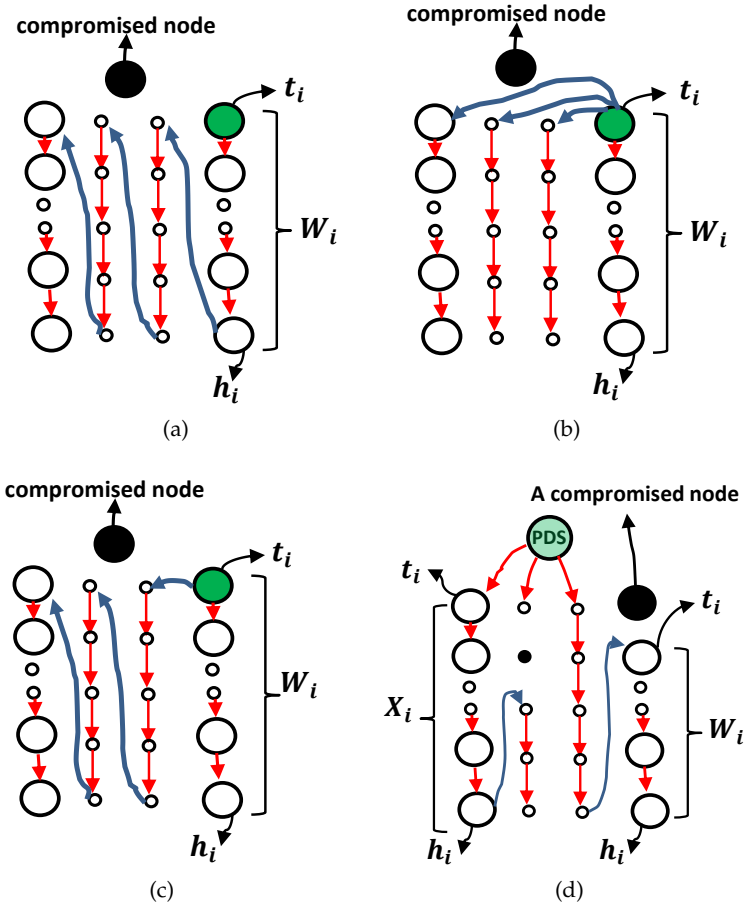


Figure 6.4: Recovering Vertices of a *Block* via Internal Blue Edges in the Presence of a *Compromised Node* ($v \in PDS$) or ($v \notin PDS$)

6.4.1 First Phase: Recovering Vertices of a Block via Internal Blue Edges in the Presence of Compromised Nodes

We seek to take the advantage of existing blue edges inside a *compromised Block* to repair the *structural controllability*. Because of the lack of the number of blue edges that exist in inside a *Block*, it is necessary to consider the external blue edges that are

incident to a *Block*. However, if both approaches are not helpful, then the addition of red edges inside a *compromised Block* is required.

Lemma 6.4 (Recovering Vertices of a Block with Compromised Nodes $v \in PDS$) *Given a* Valid Colouring *for a* Directed PDS *in* $G^{'}$. *Assuming that a* Block *has a* compromised node $v \in PDS$ *and there exists internal blue edges, then a blue edge can be coloured to red edge if it satisfies:*

1. *There is a blue edge that is incident from each $h_i(v)$ to each $t_{i+1}(v)$ then $t_i(v)$ that has no in-edge should be in a PDS in a* Block *(see Subfig 6.4.(a)).*
2. *There is a blue edge which is incident from any $t_i(v)$ to each $t_{i+1}(v)$ then $t_i(v)$ that has no in-edge should be in a PDS in a* Block *(see Subfig 6.4.(b)).*
3. *There is a blue edge that is incident from any $t_i(v)$ to exactly one (or all) $t_{i+1}(v)$ and/or a blue edge that is incident from $h_i(v)$ to the remaining $t(v)$ that is not covered yet then $t_i(v)$, that has no in-edge, should be in a PDS in a* Block, *Subfig 6.4.(c).*
4. *Otherwise, applying any previous condition without placing a PDS and go to the second phase.*

Proof. Let $G^{'} = (V, E)$ be a directed graph, satisfying the assumptions (1,2 and 3) as shown in Subsection 1.4.2 in Chapter 1. Assume that a *Valid Colouring* for a *Directed PDS* in $G^{'}$ and a *compromised node* $v \in PDS$ are given as shown in Subfig 6.4.a. According to Lemma 5.2, For each $u \in G^{'}(V)$ except vertices in PDS, there is exactly one directed path (p) from the vertices in v to u such that all the edges in p are coloured by red. We call this path as *Red-Connected Component*,denoted by X, as defined in 6.4. Therefore, if a vertex v has been compromised, then all a set of *Red-Connected Components* will be partitioned. According to the constraints in Definition 4.6, each vertex in $G^{'}(V) \setminus v \in PDS$ has at most one red in-edge and one red out-edge except head vertices in each X that have **no** red out-edges. Hence, it is not allowed to employ red edges in each X in order to recover vertices in a *Block*. Thus, only blue edges are able to control all *Red-Connected Components* partitioned from a *compromised node* $v \in PDS$. Note that tail vertices t_i in each X have **no** red in-edges incident to the tails after v has been compromised. Moreover, head vertices h_i in

each X have **no** red out-edges incident to another vertex as well. Consequently, the only blue edges that are incident from $h_i \to t_i$ or $t_i \to h_i$ can power dominated each X by applying the constraints in this lemma.

Now we introduce the recovering vertices of a *Block* in the case of a *compromised node* is $v \notin PDS$. The proof of the following lemma is similar to the proof of Lemma 6.4, and we skip it here.

Lemma 6.5 (Recovering Vertices of a Block with Compromised Nodes $v \notin PDS$) *Given a* Valid Colouring *for a* Directed PDS *in* $G^{'}$. *Supposing that a* Block *has a compromised* $v \notin PDS$ *and internal blue edges, then a blue edge is coloured to red if there is a blue edge that is incident from* $h_i(v) \in X_i$ *to* $t_i(v) \in W_i$, *where* W *denotes the* dependency path *that is leftover of a* compromised node, *or from* $u \in PDS$ *to* $t_i(v) \in W_i$ *(see Subfig 6.4.(d)); otherwise go to the second phase.*

We give this result, in particular (Lemmata 6.4 and 6.5), in a constructive form:

Algorithm 6.1: Recovering Vertices of a *Block* via Internal Blue Edges in the Presence of *Compromised Nodes*

Input : Given a *Valid Colouring* of a *Block Cut-Vertex Tree* (T^B) of a Directed PDS for $G^{'}$
Output: Recovering *Structural Controllability* via internal blue edges in a compromised *Block*

```
if exists a compromised node v ∈ PDS and a Block has internal blue edges then
    if the number of S in a Block > 1 then
        if there is e_b incident from each h_i(v) to each t_{i+1}(v) then
            Colouring e_b → e_r and d⁻t_i(v) = 0 is PDS in B^c
        else if If there is e_b incident from any t_i(v) to each t_{i+1}(v) then
            Colouring e_b → e_r and d⁻t_i(v) = 0 is PDS in B^c
        else if satisfies the constraint (3) in the definition (6.4) then
            Colouring e_b → e_r and d⁻t_i(v) = 0 is PDS in B^c
        else
            Apply any previous condition without placing a PDS and go to
            the next phase.
    else if the number of S is exactly one then
        t_i(v) ∈ W is PDS in B^c
else if exists a compromised node v ∉ PDS and a Block has internal blue edges
then
    Applying the constraint in lemma (6.5)
```

6.4.2 Second Phase: Recovering Vertices of a Block via External Blue Edges in the Presence of Compromised Nodes

If (some/all) vertices are still not controlled in the presence of *compromised nodes* via internal blue edges, then we recover *structural controllability* via the external blue edges of a *Block* (B^{ex}), that are incident to a *compromised Block* (B^{c}) including a *compromised node* (or vertex).

The proof of the following lemma is similar to the proof of Lemma 6.4 with taking into consideration that blue edges mentioned in the following lemma are incident from an external *Block* and **not** from a *compromised Block* as stated in Lemma 6.4 for the case of a *compromised node* either $v \in PDS$ or $v \notin PDS$ $v \in PDS$. Therefore, we skip the proof here.

Lemma 6.6 (Colouring an External Blue Edge to a Red Edge in a Block) *Given a* Valid Colouring *for a* Directed PDS *in* $G^{'}$. *Supposing that a* Block *has a* compromised node *either* $v \in PDS$ *or* $v \notin PDS$ *and blue edges that are incident from* B^{ex} *to* B^{c}, *then a blue edge can be coloured to red edge if it satisfies (see Figure 6.5):*

1. *There is a blue edge incident from* $v \in PDS$ *in* B^{ex} *to* $t_i(v) \in W_i$ *in* B^{c}, *and/or*
2. *there is a blue edge that is incident from* $h_i(v) \in X_i$ *in* B^{ex} *to* $t_i(v) \in W_i$ *in* B^{c}.

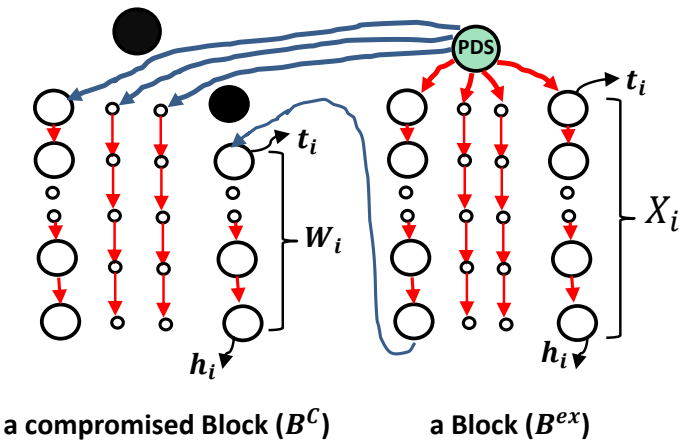


Figure 6.5: Recovering Vertices of a *Block* via External Blue Edges in the Presence of *Compromised Nodes* ($v \in PDS$) or ($v \notin PDS$)

We give this result of second phase, in particular (Lemma 6.6) in a constructive form in the following algorithm:

Algorithm 6.2: Recovering Vertices of a *Block* via External Blue Edges in the Presence of *Compromised Nodes*

Input : Given a *Valid Colouring* of a *Block Cut-Vertex Tree* (T^B) for a Directed PDS in G'

Output: Recovering *Structural Controllability* via external blue edges of a Block

1 **if** *exists compromised nodes that are not covered yet in the Algorithm (6.1)* **then**
2 **if** *there is* e_b *incident from* $B^{ex}(v) \in PDS$ *to* $t_i(v) \in W_i$ *in* B^c **then**
3 Colouring $e_b \rightarrow e_r$
4 **else if** *there is* e_b *incident from* $h_i(v) \in X_i$ *in* B^{ex} *to* $t_i(v) \in W_i$ *in* B^c **then**
5 Colouring $e_b \rightarrow e_r$
6 **else**
7 compromised nodes are still not covered yet, then apply the algorithm 6.3

6.4.3 Third Phase: Recovering Vertices of a Block via Adding Edges in the Presence of Compromised Nodes

Because of the lack of blue edges that reside in B^c or are incident from B^{ex}, moreover, each red edge are already in use to control other vertices, we identify the criteria of adding red edges inside a *compromised Block*.

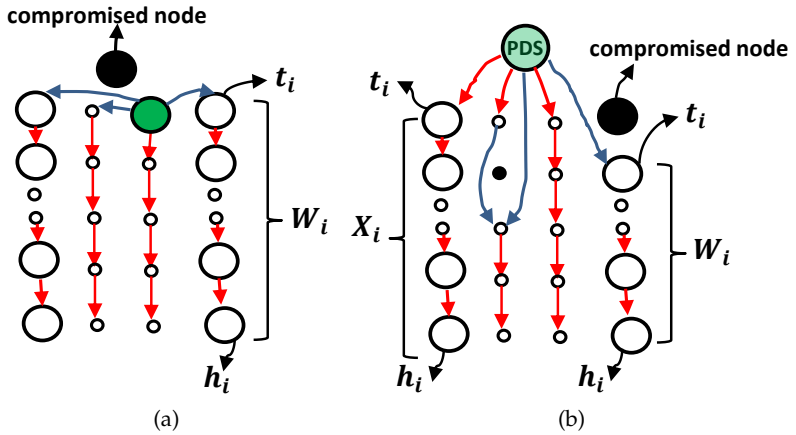


Figure 6.6: Recovering Vertices of a *Block* via Adding Red Edges in the Presence of a *Compromised Node* ($v \in PDS$) or ($v \notin PDS$)

Lemma 6.7 (The Addition of Red Edge in a Compromised Block)
Given a Valid Colouring *for a* Directed PDS *in* $G^{'}$. *Supposing that a* Block *has a* compromised node *and there are no blue edges whether to reside in* B^{c} *or to be incident from* B^{ex} *to* B^{c}, *then each* W_i *should be controlled by a red edge such that (see Figure 6.6):*

1. *if* $v \in PDS$ *is a compromised, then a red edge is added from* $t_i(v)$ *to each* $t_{i+1}(v)$, *or from each* $h_i(v)$ *except one head vertex to each* $t_i(v)$ *except one tail vertex, and place* $d^{-}t_i(v) = 0$ *or* $d^{-}_{e^r}t_i(v) = 0$ *in a PDS (see Subfig 6.6.(a)).*
2. *if* $v \notin PDS$ *is a compromised, then a red edge is added from* $v \in PDS$ *to* $t_i(v) \in W_i$, *or from* $h_i \in X_i$ *to* $t_i(v) \in W_i$ *(see Subfig 6.6.(b)).*

Proof. Let $G^{'} = (V, E)$ be a directed graph, satisfying the assumptions (1,2 and 3) as shown in Subsection 1.4.2 in Chapter 1. Assume that a *Valid Colouring* for a *Directed PDS* in $G^{'}$ and a *compromised node* $v \in PDS$ or $v \notin PDS$ are given.
We claim that if a *compromised node* is $v \in PDS$, then a red edge can be assigned from any tail $t_i(v)$ of a *Red-Connected Component*, denoted by X, to each $t_{i+1}(v)$ of X, and place $d^{-}t_i(v) = 0$ or $d^{-}_{e^r}t_i(v) = 0$ in a PDS in a *compromised Block* as shown in Subfig 6.6.(a). According to the rules of *controllability* defined in Definition 2.16, only vertices in a PDS can have more than one out-edge incident from $v \in PDS$ to any vertex in a graph. Therefore, one can place a red edge to a vertex $u \notin PDS$ that has *no* a red in-edge, provided a vertex that a red edge is coming from should be in a PDS. It can be seen that the only tails of a set of *Red-Connected Components*,denoted by S, have *no* red in-edges after $v \in PDS$ is compromised.

Now suppose for the sake of the contradiction that u is **no** a tail or head and a red edge is added from u to any tails of S. Then, u should be in a PDS as it already controls another vertex at the same time, and the number of red edges required to place will increase as a red in-edge that is incident to u should be coloured to blue in order to avoid breaching the constraints of a *Valid Colouring* as set forth herein Definition 4.6. Therefore, red edges should be added from a tail vertex to each tails of S as stated in the first constraint of this lemma. Now let a *compromised node* $v \notin PDS$. The same argument is applied when $v \notin PDS$ is a *compromised node* with

taking into consideration that a red edge is added from $v \in PDS$ to $t_i(v) \in W_i$, or from $h_i \in X_i$ to $t_i(v) \in W_i$ (see Subfig 6.6.(b)).

We give the result of third phase, in particular (Lemma 6.7) in a constructive form in the following algorithm:

Algorithm 6.3: Recovering Vertices of a *Block* via Adding Red Edges Inside a *Block* in the Presence of *Compromised Nodes*

Input : Given a *Valid Colouring* of a *Block Cut-Vertex Tree* (T^B) for a Directed PDS in $G^{'}$

Output: Recovering *Structural Controllability* via adding red edges in a compromised Block

```
if exists compromised nodes that are not covered yet in the Algorithms (6.1) and (6.2) then
    if a compromised node v ∈ PDS then
        Adding a red edge from any t_i(v) to each t_{i+1}(v);
        Placing d^-t_i(v) = 0 or d^-_{e^r} t_i(v) = 0 is PDS in B^c
    else
        Adding e^r from v ∈ PDS to each t_i(v) ∈ W_i or from h_i ∈ X_i to
        t_i(v) ∈ W_i
```

The motivation of the algorithm is not only reconstruction of the structure of a PDS in the presence of *compromised nodes* but also the ability to minimise their numbers while observing the entire system.

Lemma 6.8 (The Number of a PDS in the Presence of a Compromised Node) *Given a* Directed PDS *of $G^{'}$, reconstructed in terms of a* Valid Colouring. *Let $G^{'}$ has a* compromised node. *The cardinality of a PDS achieved by the proposed algorithm remains the same as (or less than) the number of a given PDS.*

Proof. We are given a *Valid Colouring* for a *Directed PDS* in $G^{'}$, satisfying the assumptions (1,2 and 3) as shown in Subsection 1.4.2 in Chapter 1. We prove that the number of a PDS after applying the proposed algorithm is less than a given PDS. By way of contradiction, assuming that a *compromised Block* has $v, u \in PDS$, and only v is a *compromised node*; then all the RCCs, denoted by S_v, that were connected to v should be now controlled (see Figure 6.5). By applying the three phases proposed, S_v is power dominated if and only if:

1. There is a blue edge that is incident from $u \in PDS$ to the tails of S_v, and/or

2. there is a blue edge that is incident from the heads of RCCs of u to the tails of S_v, and/or
3. by adding red edges from u to the tails of S_v.

Hence, the cardinality of a PDS will become one (i.e. $u \in PDS$), which is a contradiction to the assumption.

Let prove the equality of a PDS. The same argument is applied; supposing that a *Block* has $u \in PDS$ and $v \notin PDS$ is compromised. Then, the vertices that are reachable from v, denoted by w_i, should be controlled (see Subfig 6.4.(d)). By applying the same constraints above, w_i is power dominated with the same of number of a PDS, so the lemma is proved.

Note that in case of $v \in PDS$ is compromised, then all the neighbours of v should be power dominated, meaning each RCC that were connected to v should have at most one red in-edge by applying the three phase propose. Hence, we prove the following lemma.

Lemma 6.9 (The Relation between Dependency Paths and the Neighbours of a PDS) *The number of* dependency paths *of* $v \in PDS$ *in each* Block *is exactly equal to the number of neighbours of* v.

Proof. Given a *Valid Colouring* for a *Directed PDS* in $G^{'}$, satisfying the assumptions (1,2 and 3) as shown in Subsection 1.4.2 in Chapter 1. Let a *Block* has only one vertex $v \in PDS$ that controls all vertices in a *Block*. By applying the first role (D1) to v, all the neighbours of v are controlled by red edges that are incident from v. By way of contradiction, supposing that the *Block* has w that is not controlled by v such that w is a neighbour of v (i.e. $e_b = vw$). Thus, either:

1. w should be controlled by $u \in PDS$ in other *Block* which contradicts the assumption that a *Block* has only one vertex in a PDS that controls all vertices in a *Block*, or
2. w is a vertex in a PDS, which also contradicts the assumption, or
3. w has a red edge that is incident from $x \notin PDS$ in the *Block* of v by applying the second role D2; thus, w is not a neighbour of v. This is a contradiction, so the lemma is proved.

The following lemma completes the proof of Theorem 6.3. The proof is similar to Lemma 6.1, and we skip it here.

Lemma 6.10 (Recovering *Controllability* in Presence of Compromised Nodes) *Given a* Valid Colouring *for a* Directed PDS *in* $G^{'}$*, one can repair the* structural controllability *of* $G^{'}$ *in the presence of* compromised nodes *by reconstructing* $G^{'}$ *as a* Block Cut-Vertex Tree.

6.5 Time Complexity

Lemma 6.11 (Time Complexity of Recovering in Presence of Compromised Nodes) *Given a* Valid Colouring *for a* Directed PDS *in* $G^{'}$ *and* compromised nodes *such as* $v \in PDS$ *or* $v \notin PDS$*, one can recover* structural controllability *of* $G^{'}$ *in the presence of* compromised nodes *in* $\mathcal{O}(nc^W)$ *time for a constant* c*, where* W *denotes* dependency paths *that are the remainder of a* compromised node.

Proof. Let $G^{'} = (V, E)$ be a directed graph, satisfying the assumptions (1,2 and 3) as shown in Subsection 1.4.2 in Chapter 1. We assume that a *Valid Colouring* for a *Directed PDS* in $G^{'}$ and a *compromised node* $v \in PDS$ or $v \notin PDS$ are given. The time complexity of the algorithm satisfies the assumption (4) as in Subsection 1.4.2 in Chapter 1, where the algorithm is classified based on three phases. The total running time of the algorithm 6.1 is to utilise blue edges existing in B^c, hence in the worst case, the most time-consuming part is to determine blue edges that are incident to each *tail* of W_i in B^c in case of $v \in PDS$ where there are (4^W) states.

For the average case, it considers only the path W_i that was leftover of a *compromised node* v in case of $v \notin PDS$ where there are $(2^{(W-1)})$ states such that there are at most $\{2^{(W-1)}.4^W\}$ states in the algorithm, where W denotes the number of *dependency paths* that are reachable from a *compromised node*. Thus, the most running time of the algorithm 6.1 is $\mathcal{O}(nc^W)$. However, the most running time of the algorithm 6.2 in the average case of a compromised $v \in PDS$ or $v \notin PDS$ is $\mathcal{O}(nc^W)$, such that either $B^{ex}(v)$ where $v \in PDS$ or the *head* of $X_i \in B^{ex}$ has an blue in-edge which is incident to W_i in B^c, where there are at most $\{2^W.2^{(W-1)}\}$ states.

On the other hand, the best case is to execute the algorithm 6.3 such that if a compromised $v \in PDS$ then a red edge is added from any $t_i(v)$ to each $t_{i+1}(v)$ or if a compromised $v \notin PDS$ then a red is added from $v \in PDS$ to $t_i(v) \in W_i$, such that the running time is $\mathcal{O}(nc^W)$, where there are at most $\{2^{(W-1)}.1^W\}$ states.

6.6 Summary

Structural controllability is a highly interesting concept for understanding vulnerabilities to attack in critical infrastructures, in which adversaries with sufficient knowledge of the network distribution can disrupt the power domination relation by compromising a subset of nodes, and thus, disconnect parts of the control graph and leave parts of a graph uncontrolled; this, however, requires the ability to recover *controllability* as fast as possible since adversaries may repeatedly attack. This chapter proposed a novel algorithm to restore a control graph as far as possible if the PDS or its dependent nodes have been partially compromised without complete re-computation. The approach is based on a BLOCK DECOMPOSITION of a directed graph, allowing us to re-construct a PDS structure by applying three phases.

Chapter 7

The Effect of Rewiring Edges on the Structural Controllability

7.1 Overview

The POWER DOMINATION problem arose in the context of monitoring electric power networks by placing as few measurement devices in the system as possible; these devices have the capability of monitoring remote elements via propagation as in rule (**D2**) (see Definition 2.16). Due to the high cost of these devices, their number should be minimised while monitoring the entire system.

This chapter studies the case of sparse Erdős-Rényi Graphs with directed control edges and seek to investigate the effect of rewiring edges on the *structural controllability* of Erdős-Rényi graphs in order to achieve a minimal PDS while keeping the total number of edges unchanged. The approach is based on a STAR DECOMPOSITION of a directed graph, allowing us to identify the number of *out-neighbours* of a PDS, and ultimately achieving of a minimal PDS. The main result is that a PDS can be minimised without changing the number of edges in $\mathcal{O}(c^{E(b)} \cdot n)$ time for a constant c, where $E(b)$ denotes blue edges.

7.2 Problem Statement and Assumption

The PDS problem provides a plan for installing monitoring devices to monitor the whole power network. Since the cost of placing a minimum-size set of these devices is rather high, this algorithm aims to minimise the set of a PDS without changing

the number of edges in a given graph. Recall that underlying this approach is the modelling of large-scale cyber-physical systems as graphs with nodes (representing physical variables, sensors, and actuators) and edges (representing physical interactions or information flows between nodes). The glass of graphs that we consider in this chapter is based on the number of assumptions as stated in Subsection 1.4.2 in Chapter 1. Let $G^{'} = (V, E)$ be a directed graph, generated by $ER(n, p)$. We assume that a *Valid Colouring* for a *Directed PDS* in $G^{'}$ is given such that the edges in $G^{'}$ are coloured by red and blue edges.

•**Input:**

Given a *Valid Colouring* for an instance of a *Directed PDS* in $G^{'}$.

•**Question:**

Can we achieve a minimal PDS without changing the total number of edges while maintaining the *structural controllability* of $G^{'}$ (*e.g.* in the context of an electric power system, can we minimise the number of sensors such as Phasor Measurement Units (PMU) by re-linking transmission lines or a communication links joining two electrical sensors or actuators while keeping the total number of links unchanged).

•**Output:**

Identifying *Directed Stars* of given $G^{'}$ and colouring each blue edge of *Blue Edge-Cut Set* that is incident to vertices in a PDS to red and then rewiring a red in-edge that is incident from $v \in PDS$ to its *open neighbours*.

7.3 Reconstructing DPDS via Directed Star Decomposition

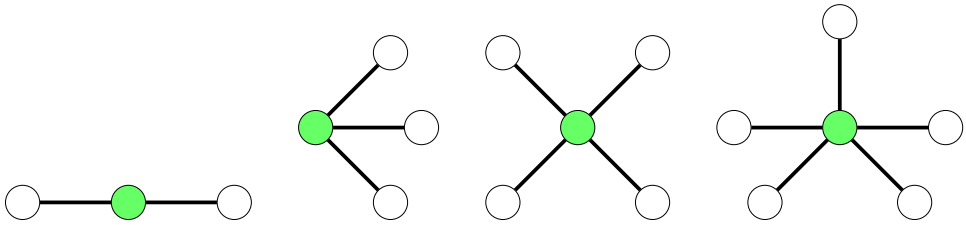
Our interest lies primarily in studying the effect of rewiring directed edges of a PDS structure in Erdős-Rényi graphs. Therefore, we seek to identify a directed star of a *Directed PDS* structure in terms of a *Valid Colouring*, allowing us to define the *open neighbours* of $v \in PDS$. Consequently, the proposed algorithm can impact on *structural controllability* of a directed graph, and therefore, minimise the number of a PDS. Note that *Directed Stars* has a different definition in terms of direction of edges compared to *Undirected Stars*. We now introduce some basic definitions:

Definition 7.1 (An Internal Vertex)

An internal vertex *in undirected graph is a vertex of degree of at least 2.*

Definition 7.2 (An Undirected Star)

A star S_k of undirected graph of order k, where k denotes a set of vertices in a star, is a tree with one internal vertex *having degree of $|k-1|$ and the set of $(k-1)$ vertices have degree of 1 (see Figure 7.1).*



(a) S_3 (b) S_4 (c) S_5 (d) S_6

Figure 7.1: Examples of Undirected Stars

Definition 7.3 (The Neighbourhoods of a Vertex)

A vertex u is called a neighbour of v if there is an edge between u and v in $G^{'}$ (i.e. $\{uv\} \in E$). The open neighbourhood *of a vertex v, denoted by $N(v)$, is the set of all neighbours of a vertex v. The* closed neighbourhood *of a vertex v, denoted by $N[v]$, is defined as $\{v\} \cup N(v)$.*

Definition 7.4 (A Dependency Path in a *Valid Colouring*)

A dependency path *(P) from u to v in a* Valid Colouring *is a sequence of vertices such that all red edges in P are directed from u to v and all blue edges are directed from v to u. Therefore, a directed path is a dependency path with only directed red edges.*

Definition 7.5 (A Set of Dependency Paths in a *Valid Colouring*)

A set of dependency paths*, denoted by P_i, in a* Valid Colouring *of a Directed PDS is more than one* dependency path *P that is incident from $v \in PDS$ to p_i, such that $\{v \to P_1, \ldots, v \to P_i\}$.*

Definition 7.6 (The Diameter of a Graph)

The distance $d_{G'}(u, v)$ between two vertices u and v is the length of shortest path joining u and v in $G^{'}$. The Diameter (D) of $G^{'}$ is the greatest distance between any pair of vertices (u, v).

Definition 7.7 (A Blue Edge-Cut Set in a *Valid Colouring*)

Given a Valid Colouring *for a* Directed PDS *in* $G^{'}$ *reconstructed in terms of red and blue edges, a* Blue Edge-Cut Set, *denoted by* Y, *of* $G^{'}$ *is a set of blue edges satisfying (see Subfig 7.2.(b)):*

1. *The removal of all blue edges in* Y *disconnects* $G^{'}$ *to sub-graphs such that all vertices in each sub-graph that are reachable from* $v \in PDS$ *by red edges (see Subfig 7.2.(c)).*
2. *The removal of some (but not all) of blue edges in* Y *does not disconnects* $G^{'}$.

Note that there may some blue edges that are not belonging to a *Blue Edge-Cut Set* since they are not satisfying the constraints in Y such as the edge $(x_{11}x_{13})$ and (x_7x_8) in Subfig 7.2.(b). Therefore, they do not consider when computing a *Blue Edge-Cut Set* in a *Valid Colouring*.

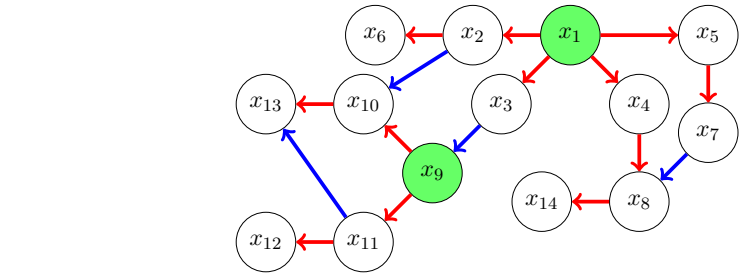
Definition 7.8 (A Directed Star (DS) in a *Valid Colouring*)

Let k *denotes a set of vertices in a star. A* Directed Star DS_k *in* Valid Colouring *of order* k *is a tree which consists of an* internal vertex v *in a PDS having a (red) out-edge of degree* $|k-1|$ *and the other vertices of* $k \setminus v \in PDS$ *have a (red) in-edge of degree of 1 (i.e. the diameter of underlying undirected graph is at most 2).*

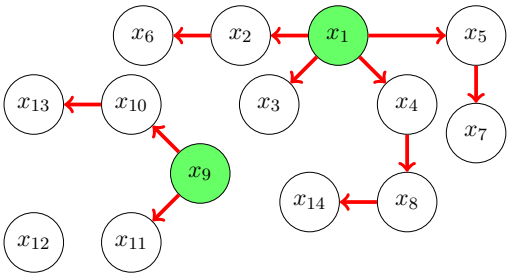
Example 6 Consider Subfig 7.2.(c), it shows a *Directed Star* where green vertices imply vertices in a PDS and white ones to a set of *open neighbours* of $v \in PDS$ such that a set of *open neighbours* of *Directed Stars* $x_1, x_9 \in PDS$ is $N_{DS}(x_1) = \{x_2, \ldots, x_5\}$ and $N_{DS}(x_9) = \{x_{10}, x_{11}\}$ respectively. However, the closed neighbours of $v \in PDS$ are $N_{DS}[x_1] = N_{DS}(x_1) \cup x_1$ and $N_{DS}[x_9] = N_{DS}(x_9) \cup x_9$, such that $N_{DS}[x_1] = \{x_1, x_2, \ldots, x_5\}$ and $N_{DS}[x_9] = \{x_9, x_{10}, x_{11}\}$ respectively. □

Definition 7.9 (A Head and Tail of a Dependency Path)

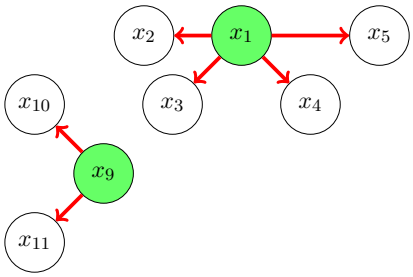
A tail vertex *of a* dependency path *(P) is defined as a vertex* u *with a red in-edge that is incident from* $v \in PDS$ *to* u *(i.e. the first vertex after* v*), and a* head vertex *is the last vertex of* P *with no red out-edge.*



(a) *Valid Colouring* of a directed graph



(b) The reomoval of Blue Edge-Cut Set



(c) Directed stars

Figure 7.2: Decomposition of *Directed Stars* in a *Valid Colouring* of Directed Graph

Example 7 Consider Subfig 7.2.(a), $x_1 \in PDS$ has four *dependency paths*. For instance, a *tail* and *head vertices* of the *dependency path* $P_2 = \{x_4, x_8, x_{14}\}$ is x_4 and x_{14}, respectively. However, in some cases, a *tail vertex* can be a *head vertex* at the same time (and vice verse) such $P_4 = \{x_3\}$ of x_1. □

7.4 The Process of Rewiring Edges

The strategy of the algorithm is based on two stages, where the first stage aims to identify *Directed Stars* of given $G^{'}$. The second stage seeks to colour each blue edge of *Blue Edge-Cut Set* Y that is incident to vertices in a PDS to red and then to rewire a red in-edge that is incident from $v \in PDS$ to its *open neighbours*.

7.4.1 The First Stage

The importance of the stage is to define a set of *open neighbours* that have power to minimise the number of a PDS in $G^{'}$ by identifying a *Directed Star* that forms a tree with its root belonging to a PDS. According to Definition 4.6 in Chapter 4, we recall that the rules of *structural controllability*:

a. It is not allowed to have two vertices sharing the same their *tail vertex* except a vertex in a PDS such vertex $x_9 \in PDS$ in Figure 7.2 has two vertices $\{x_{10}, x_{11}\}$ sharing the same their *tail vertex* (x_9).
b. Each vertex has at most one red in-edge and/or at most one red out-edge such as x_{10}, x_{11} except a vertex in a PDS such as x_9 in Figure 7.2.

Therefore, it is required to find a way to control a set of *open neighbours* instead of edges that are incident from a vertex in a PDS in order to affect the *structural controllability* of Erdős-Rényi graphs, and thus, to minimise the number of a PDS.

Lemma 7.1 (The Identification of a Directed Star in a *Valid Colouring*)
Given a Valid Colouring *for a directed graph $G^{'}$, one can identify a* Directed Star *by:*
a. Partitioning $G^{'}$ into sub-graphs through removing a Blue Edge-Cut Set *Y of $G^{'}$, and*
b. identifying vertices in a PDS and its open neighbours, *then*
c. decomposing a set of closed neighbours of vertices in a PDS, (i.e. ignoring all vertices of dependency paths *except* open neighbours *of vertices in a PDS).*

Proof. Let $G^{'} = (V, E)$ be a directed graph satisfying the assumptions (1,2 and 3) as shown in Subsection 1.4.2 in Chapter 1. Assume a *Valid Colouring* for $G^{'}$ is given (*e.g.* Figure 7.2), where $G^{'}$ has two PDS(s) $\{x_1, x_9\}$ and each vertex in a PDS has more than one *dependency path* that are incident from x_1 and x_9 such that x_1 has $P_1 = \{x_2, x_6\}$, $P_2 = \{x_4, x_8, x_{14}\}$, $P_3 = \{x_5, x_7\}$ and $P_4 = \{x_3\}$, while x_9 has $P_1 = \{x_{10}, x_{13}\}$ and $P_2 = \{x_{11}, x_{12}\}$. However, by eliminating a set of *Blue Edge-Cut* that is a part of connectivity but does not affect the *controllability* of $G^{'}$, the graph will be partitioned into sub-graphs such that each block is reachable from x_1 and x_9 by red edges as shown in Subfig 7.2.(b). The last step is to decompose each sub-graph into a *Directed Star* by only considering vertices in a PDS such x_1, x_9 and its *open neighbours* $N_{DS}(x_1)$ and $N_{DS}(x_9)$. According to Definitions 7.3, 7.6 and 7.8, the diameter of a *Directed Star* should be at most 2 as illustrated in Subfig 7.2.(c). Hence, *Directed Stars* of $G^{'}$ can be a set of closed neighbours of x_1 and x_9 such that
a. $N_{G'}[x_1] = N_{G'}(x_1) \cup x_1 \Rightarrow \{x_1, x_2, x_3, x_4, x_5\}$.
b. $N_{G'}[x_9] = N_{G'}(x_9) \cup x_9 \Rightarrow \{x_9, x_{10}, x_{11}\}$.

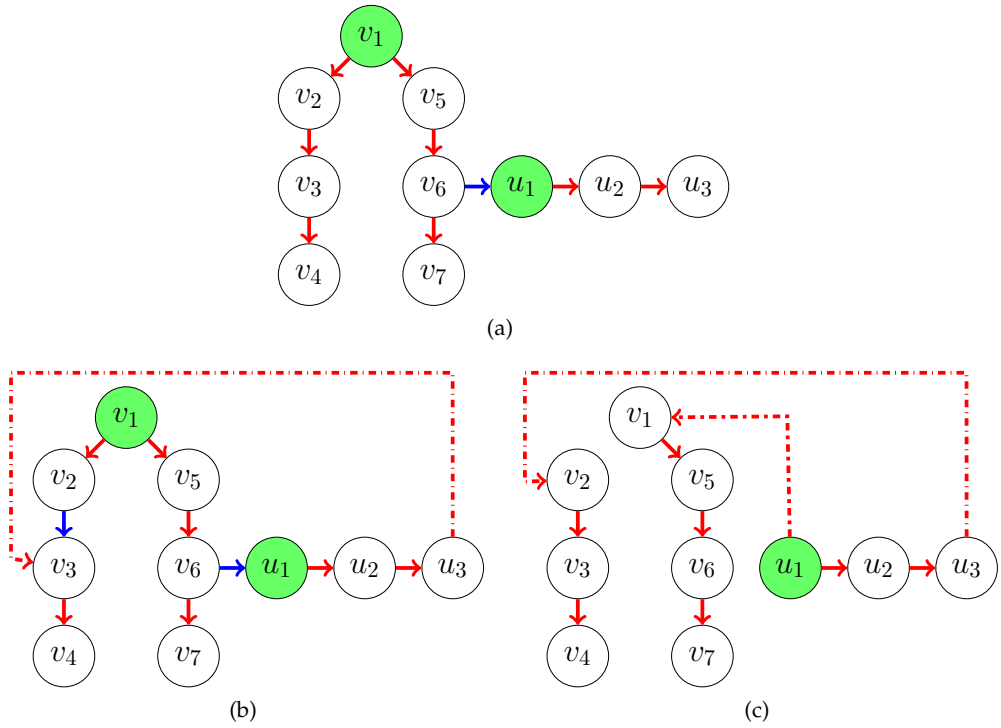


Figure 7.3: A Directed Graph

Theorem 7.2 (The Impact of a Open Neighbours Set on Vertices in a PDS)
A set of open neighbours *of vertices in a PDS has power to minimise the number of a PDS in a directed graph.*

Proof. We are given a *Valid Colouring* for a *Directed PDS* in $G^{'}$ that satisfies the assumptions (1,2 and 3) as stated in Subsection 1.4.2 in Chapter 1. Suppose $v, u \in PDS$ where each vertex has *dependency paths*. Therefore, we prove this theorem by two ways:

a. We show that a vertex, which is not a set of *open neighbours* of $v, u \in PDS$, will not change the number of a PDS. By Definition 7.3, each vertex in $G^{'}$ except a PDS should controlled by only one red in-edge. Therefore, if $w \notin N(v) \cup N(u)$ is controlled by z as $d^{+}_{E_r}(z) = 0$ (i.e. has no red out-edge) such that a red edge is incident from $z \rightarrow w$, then the number of a PDS is still the same as w is not in a set of *open neighbours* of v and u (i.e. not a vertex-disjoint with the *tail* endpoints in a PDS) (see Subfig 7.3.b).

b. Now assuming that $w \in N(v)$ is controlled by z such that there is a red edge incident from $z \rightarrow w$. Therefore, by assigning a red edge from u to v, then all vertices in $G^{'}$ are controlled by one vertex $u \in PDS$ (see Subfig 7.3.c).

7.4.2 The Second Stage

The first mechanism is based colouring a blue edge to red as in Lemma 7.3, and the other focuses on redirecting blue edges coloured to red as in Lemma 7.4:

7.4.2.1 Colouring and Re-directing Blue Edges Mechanism

As the proposed algorithm is based on a *Valid Colouring* in terms of a set of blue and red edges set forth in Definition 4.6, the following lemmata re-formulate important constraints for colouring and re-directing blue edges as follows:

Lemma 7.3 (Colouring Blue Edges in a *Valid Colouring*)

Given a Valid Colouring *for a* Directed PDS *in* $G^{'}$*, a blue edge is coloured to red if:*

a. There exists (at least) one blue edge $e^b(vu)$ *that is incident from a vertex* $v \notin PDS$ *to a vertex* $u \in PDS$*, or*

b. there exists (at least) one blue edge $e^b(vw)$ *that is incident from a vertex* $v \notin PDS$ *to a vertex* $w \notin PDS$*.*

Proof. Given a *Valid Colouring* for a *Directed PDS* in $G^{'}$, satisfying the assumptions (1,2 and 3) as stated in Subsection 1.4.2 in Chapter 1. Assume $e^b(vu)$ is a blue edge that is incident from a vertex $v \notin PDS$ to a vertex $u \in PDS$. Then by applying a *Valid Colouring* in Definition 4.6 a vertex v must **not** have a red in-edge incident to it, but v may have a blue in-edge $e^b(vu)$. Therefore, this blue edge can be used to control a vertex in a PDS (e.g. u) by colouring $e^b(vu)$ to red but; however, u will no longer be a vertex in a PDS in order to avoid violating the constraints of *structural controllability* as stated in Definition 2.16. When $e^b(vu)$ is coloured to red, we can apply *Rewiring Red Edges* mechanism to u in order to meet the constraints of a *Valid Colouring* approach as defined in Definition 4.6 (see Figure 7.4). Hence, we can colour a blue edge $e^b(vu)$ to red as mentioned in the first constraint of this lemma.

The same argument is applying to prove the second constraint of this lemma. Assume that a blue edge $e^b(vw)$ is incident from a vertex $v \notin PDS$ to a vertex $w \notin PDS$; for this case a vertex w is already power dominated by a vertex z by a red out-edge that is incident from z to w. Therefore, $e^b(vw)$ can be coloured to red in order to control a vertex $u \in PDS$, provided this blue edge should redirected from a vertex x which has **no** red in-edge, and *Rewiring Red edges* mechanism should be applied to u as shown in Figure 7.6.

The following lemma completes the proof of Lemma 7.3. The proof is similar to Lemma 7.3, and we skip it here.

Lemma 7.4 (Re-directing Blue Edges in a *Valid Colouring*)
Given a Valid Colouring *for a* Directed PDS *in* $G^{'}$*, a blue edge can be re-directed if and only if:*

a. *There exists a blue out-edge that is incident from* $v \notin PDS$ *to a vertex* $u \in PDS$*, provided* v *has a red out-edge which is incident to any vertex at the same time.*
b. *There exists a blue out-edge that is incident from* $v \notin PDS$ *to a vertex* $u \notin PDS$*, provided* v *has a red out-edge which is incident to any vertex at the same time.*
c. *There exists a blue out-edge that is incident from* $v \notin PDS$ *to a vertex* $u \notin PDS$*, provided* v *has* ***no*** *a red out-edge.*

Now we illustrate the constraints mentioned in Lemmata 7.3 and 7.4:

Definition 7.10 (The Case Enumeration for Colouring and Re-directing Blue Edges)
Given a Valid Colouring *for a* Directed PDS *in* $G^{'}$*, Colouring and Re-directing blue edges can be achieved by applying an exhaustive search to cover all the possible cases as follows:*

Case (1): Consider Figure 7.4, according to Lemma 7.3, the only edge that should be coloured to red is a blue edge that is incident from a vertex $v \notin PDS$ to a vertex $u \in PDS$, provided there is **no** a red out-edge which is incident from v to any other vertex (see Subfig 7.4.(a)). Note that a vertex v with the blue edge $e^b(v_5u_1)$, when applying *Rewiring Red edges* mechanism (we will explain that in the following section) to a vertex u_1, v_5 must have **no** red out-edge incident to any vertex, otherwise

it may lead to breach the rules of *structural controllability*, and therefore, disrupt the control of a graph (an example of this in Case (2)).

As v_5 with the edge $e^b(v_5u_1)$ has no red out-edge, this edge should be coloured to red with taking into consideration for applying *Rewiring Red edges* mechanism (elaborated in Lemma 7.5 as shown in Subfig 7.4.(b). Note that *Rewiring Red edges* in this case can result in:

(a.) $u_1 \rightarrow u_2 \rightarrow u_3 \rightarrow u_4 \rightarrow u_5$, or

(b.) $u_1 \rightarrow u_4 \rightarrow u_5 \rightarrow u_2 \rightarrow u_3$.

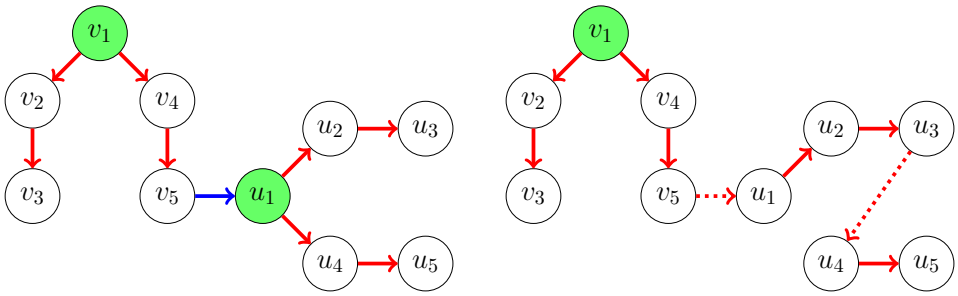


Figure 7.4: The Case (1) of Colouring a Blue Edge in a *Valid Colouring*

Case (2): Assuming that a vertex v_5 has a red out-edge which is incident to any vertex in addition to a blue out-edge that is incident to $u \in PDS$ as illustrated in Subfig 7.5.(a). By applying *Rewiring Red edges* mechanism to vertex u_1, a blue edge $e^b(v_5u_1)$ should be coloured to red (according to Lemma 7.3), however, at the same time vertex v_5 has a red out-edge which is incident to vertex v_6. This means v_5 should be in a PDS because it controls more than one vertex at the same time as set forth in Definition 4.6. So, vertex u_1 will no longer in a PDS as shown in Subfig 7.5.(b). Consequently, the number of a PDS of directed graph in Subfig 7.5.(b), after applying *Rewiring Red edges* mechanism is still the same number with a slight difference in a set of vertices $\{v_1, v_5\}$ instead of $\{v_1, u_1\}$. Therefore, *Rewiring Red edges* mechanism has no effect on the *structural controllability* a directed graph as shown in Subfig 7.5.(b). However, to obtain the minimisation of a PDS in this case, it should apply redirecting a blue edge in Lemma 7.4) to this blue out-edge $e^b(v_5u_1)$ satisfying:

a. The direction of a blue edge should be re-directed from a vertex that has no red out-edge such as v_6 or v_3 to a vertex u_1 such that $e^b(v_6u_1)$ or $e^b(v_3u_1)$.
b. A blue edge should be coloured to red as in Subfig 7.5.(c) where $e^r(v_6u_1)$.
c. *Rewiring Red edges* mechanism should be applied to a vertex u with a blue edge incident to it such as u_1 (see Subfig 7.5.(c)).

Therefore, a vertex v_1 power dominates all vertices of a directed graph that results in the minimisation of the number of a PDS to one vertex. Note that there is no change in applying *Rewiring Red edges* mechanism to a vertex u_1 for each case, however, the only difference is how to rewire a blue edge that is incident to a vertex $u_1 \in PDS$ while having a red out-edge which is incident to another vertex.

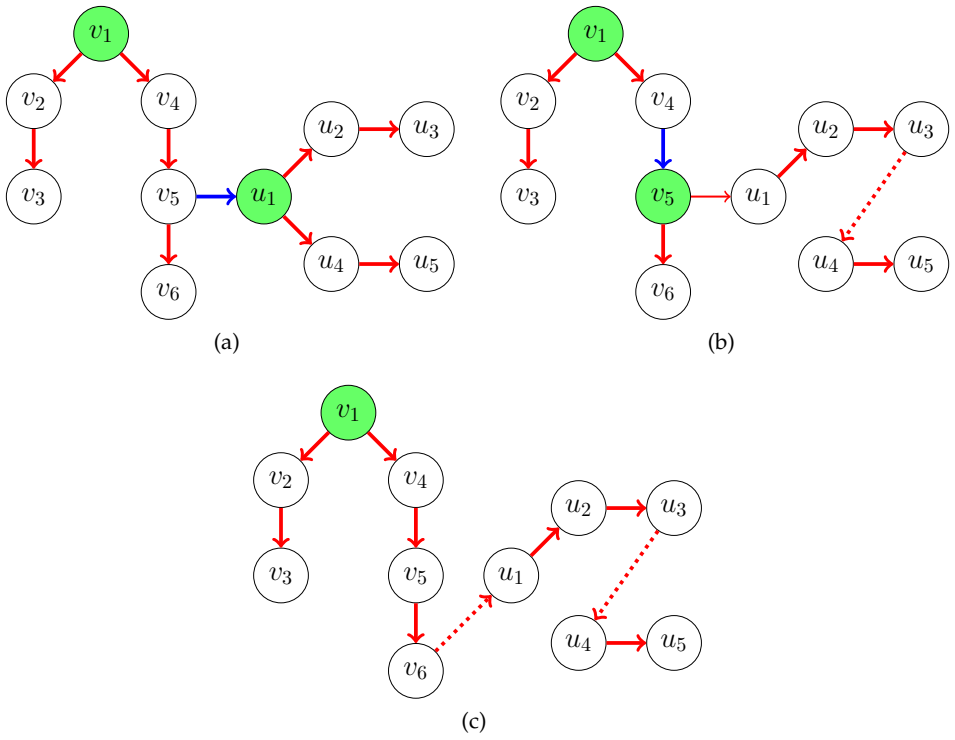


Figure 7.5: The Case (2) of Colouring a Blue Edge in a *Valid Colouring*

Case (3): Supposing that a vertex v_5 has a red out-edge which is incident to any vertex in addition to a blue out-edge that is incident to $u \notin PDS$ as illustrated in Subfig 7.6.(a). It can be seen when colouring the blue out-edge $e^b(v_5u_2)$ to red, after that applying *Rewiring Red edges* mechanism to the vertex u_1; thus, u_1 is no longer in a PDS and the red out-edge $e^r(u_1u_2)$ should be re-directed from a vertex u_3 to u_1

as shown in Subfig 7.6.(b). Therefore, there is no significant effect on the number of a PDS. However, to obtain a positive result, it should rewire redirect a blue edge from a vertex with no red out-edge to a vertex in a PDS such as $v_6 \rightarrow u_1$, and then colour it to red as illustrated in Subfig 7.6.(c). Consequently, *Rewiring Red edges* mechanism can be applied to vertex u_1 in order to affect the number of a PDS (see Subfig 7.6.(c)).

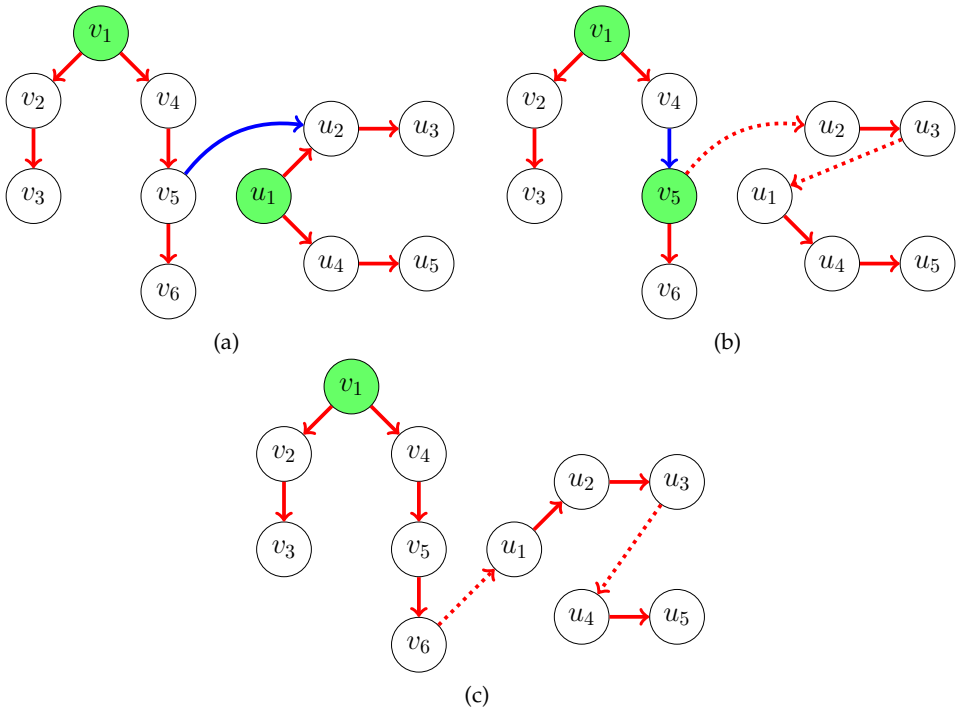


Figure 7.6: The Case (3) of Colouring a Blue Edge in a *Valid Colouring*

Case (4): On the other hand, assuming that a blue out-edge which is incident to $u \notin PDS$ and there is no red-out-edge that is incident from v_5 to any vertex as shown in Subfig 7.7.(a). Thus, as long as there is no red out-edge sharing the same *tail vertex* with the blue edge, then $e^b(v_5u_2)$ should be re-directed from v_5 to u_1 and coloured to red while applying *Rewiring Red edges* mechanism to vertex u_1 in order to minimise the number of a PDS as represented in Subfig 7.7.(b). Note that this case is similar to case (1) in terms of applying *Rewiring Red edges* mechanism to u_1, but with a difference in how rewiring a blue out-edge is applied.

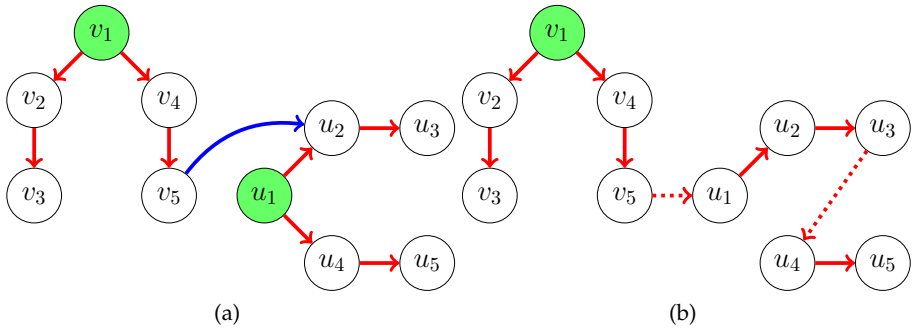


Figure 7.7: The Case (4) of Colouring a Blue Edge in a *Valid Colouring*

We give the result of Lemmata 7.3 and 7.4 in a constructive form in the following algorithm:

Algorithm 7.1: Colouring and Redirecting Blue Edges in Rewiring Red Edges Mechanism

Input : Given a directed PDS of $G^{'}$, reconstructed in terms of *Valid Colouring*
Output: Colouring and Redirecting blue edges to red edges

A blue edge $e^b = vu$ is redirected and coloured to red if and only if satisfying:;
if $v \notin PDS$ *is incident to* $u \in PDS$, *provided v has a red out-edge* **then**
    $e^b = vu \Rightarrow Red$
else if $v \notin PDS$ *is incident to* $u \in PDS$, *provided v has* ***no*** *a red out-edge* **then**
    $e^b = vu \Rightarrow Red$
else if $v \notin PDS$ *is incident to* $u \notin PDS$, *provided v has a red out-edge* **then**
    $e^b = vu \Rightarrow Red$
else if $v \notin PDS$ *is incident to* $u \notin PDS$, *provided v has* ***no*** *a red out-edge* **then**
    $e^b = vu \Rightarrow Red$
else
    Go to Subsection (7.4.3)

7.4.2.2 Rewiring Red Edges Mechanism

Together with the Lemmata 7.3 and 7.4, this third mechanism that identifies the important constraints on how to obtain *Rewiring Red Edges*, denoted by (RRE). As proved in Theorem 7.2, the *open neighbours* of a PDS have an ability to impact on structural controllability of Erdős-Rényi graphs, in particular directed graphs.

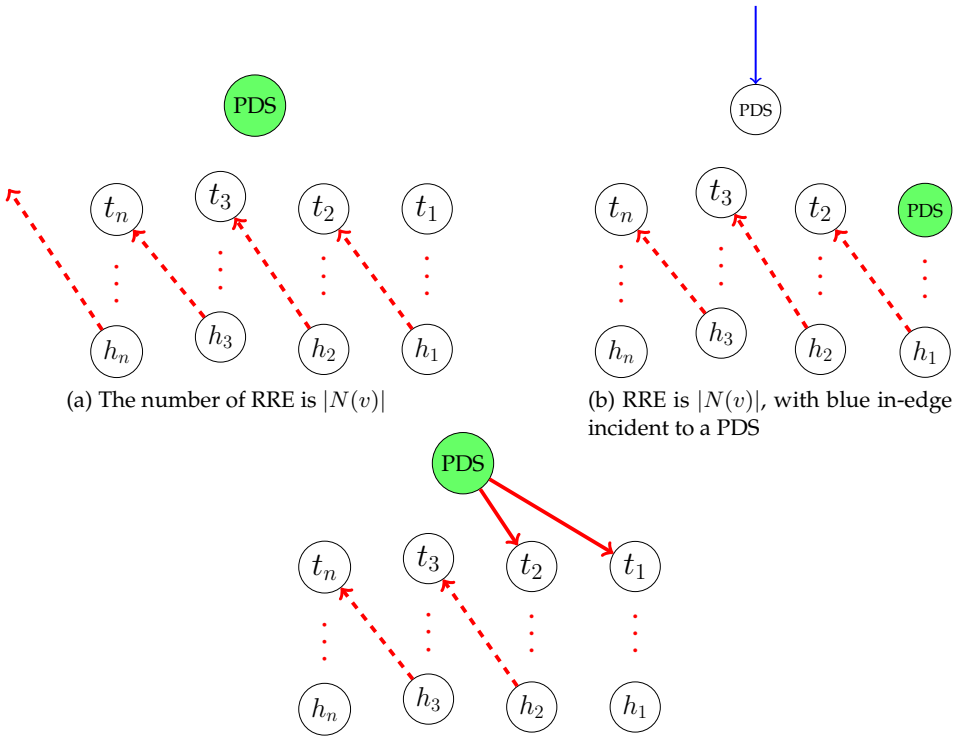
Lemma 7.5 (The Number of Rewiring Red Edges (RRE))

Let $v \in PDS$ be an internal vertex *of a* Directed Star *in a* Valid Colouring*. The number of red edges required to rewire is equal to the number of* open neighbours $N(v)$ *except one red edge such that RRE is* $|N(v) - 1|$.

Proof. We are given a *Valid Colouring* for a *Directed PDS* in $G^{'}$, satisfying the assumptions (1,2 and 3) as stated in Subsection 1.4.2 in Chapter 1. Given a *Directed Star* with an *internal vertex* $v \in PDS$ where its out-degree of 4. We show that the number of red edges that should be re-directed is equal to the number of *open neighbours* of v except one red edge by two cases.

Case (1): Assuming that the number of red edge required to rewire is exactly the same of the number of $N(v)$ where RRE is $|N(v)|$ and a vertex v in a PDS has no blue in-edge as shown in Subfig 7.8.(a). When *Rewiring Red Edges* that are incident from v to its $N(v)$, it results in a vertex $v \in PDS$ will be disconnected from a *Directed Star* which contradicts with the given assumption, moreover, a vertex in a PDS will be isolated from a directed graph (i.e. there is no directed edge incident to it). However, the same argument is applied to the case of a vertex $v \in PDS$ has a blue in-edge as shown in Subfig 7.8.(b). By applying *Rewiring Red Edges*, a vertex with no red in-edge will be in a PDS and a vertex that was in a PDS with a blue in-edge will be controlled by a vertex that is pointed to it. Therefore, there is no difference in the number of a PDS as it has been replaced by another vertex.

Case (2): Now let the number of red edges is equal to the number of $|N(v) - 2|$ as shown in Subfig 7.8.(c). So, the number of red edges is equal to $(|4 - 2|)$ where there is only two red edges required to be rewired. Therefore, the remaining red edges are still sharing the same *tail vertex* $v \in PDS$, meaning it cannot obtain the minimisation of a PDS, and therefore, there is no point of *Rewiring Red Edges*. Therefore, the number of red edges required to rewire is equal to the number of *open neighbours* $N(v)$ except one red edge such that RRE is equal to $|N(v) - 1|$ (see Subfig 7.9.(c)). So, the lemma is proved.



(a) The number of RRE is $|N(v)|$

(b) RRE is $|N(v)|$, with blue in-edge incident to a PDS

(c) RRE is $|N(v) - 2|$

Figure 7.8: An Example of Identification of the Number of Rewiring Red Edges in a *Valid Colouring*

Together with the Lemma 7.5, we define the mechanism of *Rewiring Red Edges*.

Lemma 7.6 (Rewiring Red Edges (RRE) Mechanism)

Given a Valid Colouring *for a* Directed PDS *in* $G^{'}$ *with an* internal vertex $v \in PDS$ *such that* $d^{+}(v) \geq 2$. *All red edges* (Z) *that are incident from* v *to vertices in a set of* open neighbours $N(v)$ *except exactly one red edge should be re-directed from* head vertices (h_n) *with no red out-edge, where* $d^{+}_{er}(h_n) = 0$, *to all* tail vertices $(|t_n - 1|)$ *except one* tail vertex *of each* dependency path *(as shown in Figure 7.9) such that*

$$h_1 \rightarrow t_2, h_2 \rightarrow t_3, \ldots, h_{n-1} \rightarrow t_n$$

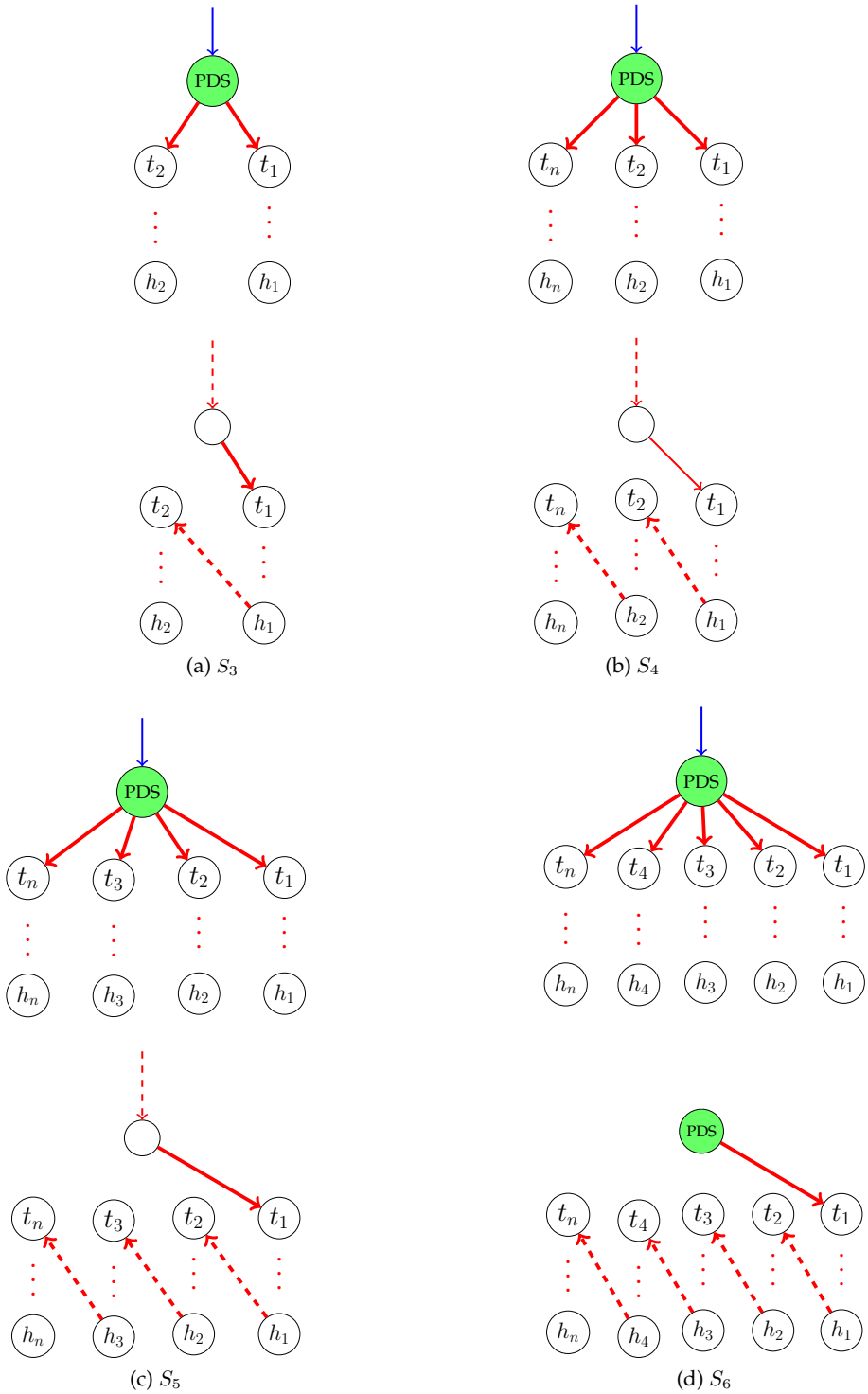
Proof. Given a *Valid Colouring* for a *Directed PDS* in $G^{'}$, satisfying the assumptions (1,2 and 3) as shown in Subsection 1.4.2 in Chapter 1. This lemma shows how *Rewiring Red Edges* mechanism is applied. Consider *Directed Stars* in Figure 7.9 with out-degree > 1; as proved in Lemma 7.5, the number of red edges required

to rewire is equal to the number of *open neighbours* of a vertex in a PDS except exactly one red edge. So, each red edge is incident to a set of *open neighbours* of a PDS except one red edge that should be re-directed from each *head vertex* to each *tail vertex* except one *tail vertex* which has a red in-edge incident from a vertex in a PDS. Note a blue edge incident to a vertex in a PDS should be coloured in order to minimise the number of a PDS by applying the first mechanism (i.e. Colouring and Re-directing Blue Edges Mechanism). However, a vertex in a PDS may have no blue in-edge, and therefore, we will explain how to solve the case in Section (7.4.3).

Theorem 7.7 (The Impact of Rewiring Edges on *Structural Controllability*) *Given a* Valid Colouring *for a* Directed PDS *in* $G^{'}$*, one can minimise the number of a PDS by rewiring (blue, red) edges while maintaining* structural controllability *of* $G^{'}$.

Proof. Let $G^{'} = (V, E)$ be a directed graph satisfying the assumptions (1,2 and 3) as stated in Subsection 1.4.2 in Chapter 1. Assume a *Valid Colouring* for a *Directed PDS* in $G^{'}$ is given. As the mechanisms above showed how to minimise the number of a PDS while maintaining *structural controllability* of a digraph, we show that applying one mechanism without the other has no effect on the control structure of a directed graph, ultimately the process of minimisation of a PDS will not be achievable.

Suppose the *Rewiring Red Edges* mechanism is applied (i.e. the only third mechanism); therefore, the only result that can be obtained is to link each *dependency path* with the other; this means instead of having more than one path connected to a vertex in a PDS, there will be exactly one path which has no change in the number of a PDS. Moreover, the *Rewiring Red Edges* mechanism concentrates on only red edges that have no power to minimise a PDS without taking advantage of blue edges as proved in Lemmata 7.3, 7.4 and 7.5 such as a directed graph in Subfig 7.9.(d), where a vertex in a PDS has no blue in-edge. Now, assume the *Rewiring Red edges* mechanism is applied, so the graph will still controlled by the same PDS with exactly one *dependency path* instead of 5 *dependency paths* by joining each *head vertex* to *tail vertex* except one *tail*. Hence, it is vital to apply all mechanisms in order to minimise a PDS while reconfiguring *structural controllability* of a directed graph.



(a) S_3 (b) S_4

(c) S_5 (d) S_6

Figure 7.9: Case Enumeration of Rewiring Red Edges Mechanism in a *Valid Colouring*

We give the result of Lemmata 7.5 and 7.6 also in a constructive form in the following algorithm:

Algorithm 7.2: Rewiring Red Edges (RRE) Mechanism

Input : Given a *Valid Colouring* for a Directed PDS of out-degree > 1 with internal vertex $v \in PDS$ in $G^{'}$

Output: Rewiring red edges

```
Z ← The number of red edges incident from v to in N(v) except one red edge;
for 1 to Z do
    i ← the number of h_n;
    j ← the number of t_n;
    for 1 to i-1 do
        for 2 to j do
            h_i → t_j    ▷ Redirecting each red edge in Z from each (h_n) with no
                           red out-edge to each (t_n − 1) of each dependency path
```

7.4.3 The Addition of Red Edges to a Vertex in a PDS

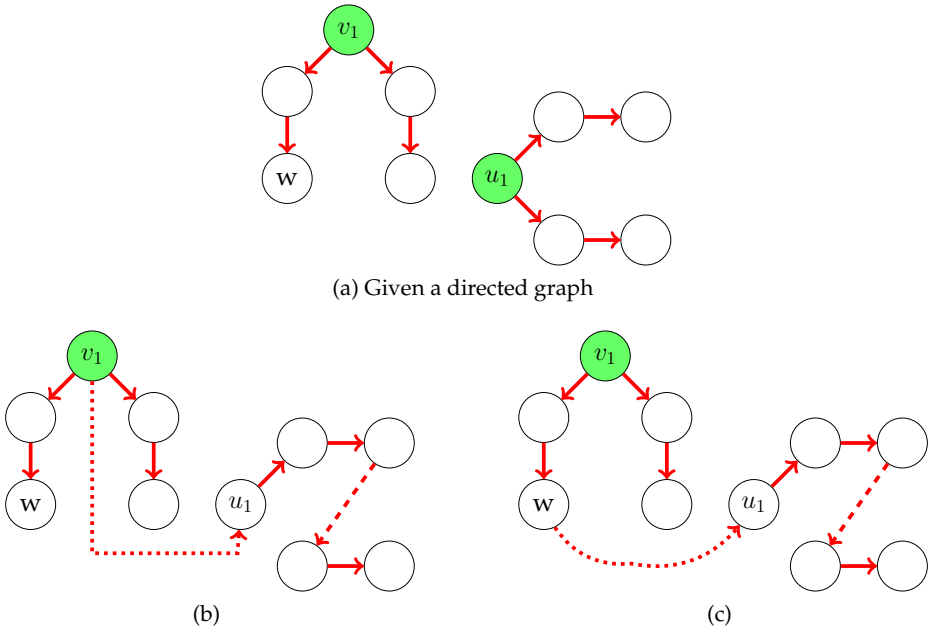
As mentioned above, the first mechanisms focus on re-directing and colouring blue edges that collaborates with *Rewiring Red Edges* mechanism to have an impact on *structural controllability* of Erdős-Rényi graphs. However, vertices in a PDS, in some cases, have no blue in-edges. Therefore, it is necessary to add a red edge to a vertex in a PDS, in case there is no blue in-edge that is incident to $v \notin PDS$, in order to minimise the number of a PDS. This can lead to change the assumption mentioned in Chapter 1 where the number of edges in a given directed graph $G^{'} = (V, E)$ will be increased.

The proof of the following lemma is similar to Lemma 6.7, and we skip it here.

Lemma 7.8 (The Addition of Red Edges in a *Valid Colouring*)
Given a Valid Colouring *for a* Directed PDS *in* $G^{'}$. *when there exist insufficient blue edges in a given* $G^{'}$, *then a red edge, as represented by dotted lines in Figure 7.10, should be added from:*

1. *A vertex* $v \in PDS$ *to* $u \in PDS$ *(see Subfig 7.10.(b)).*
2. *A vertex* $w \notin PDS$ *to* u, *provided* w *has* ***no*** *a red out-edge (see Subfig 7.10.(c)).*

Note that once a red edge has been added as defined in Lemma 7.8, then *Rewiring Red Edges* mechanism should be applied to u_1, as represented by dashed lines in Figure 7.10 , according to Definition 7.6.



(a) Given a directed graph

(b) (c)

Figure 7.10: Case Enumeration for Adding Red Edges in a *Valid Colouring*

7.5 Time Complexity

Lemma 7.9 (Time Complexity of Rewiring Edges in *Structural Controllability*) *Given a* Valid Colouring *for a* Directed PDS *in* $G^{'}$ *constructed as* Directed Stars, *one can impact on the* structural controllability *of* $G^{'}$ *to minimise the number of a PDS while keeping the number of edges unchanged in* $\mathcal{O}(c^{E(b)} \cdot n)$ *time for a constant* c, *where* $E(b)$ *denotes blue edges.*

Proof. Let $G^{'} = (V, E)$ be a directed graph, satisfying the assumptions (1,2 and 3) as shown in Subsection 1.4.2 in Chapter 1. We assume that a *Valid Colouring* for a *Directed PDS* in $G^{'}$ is given and constructed as *Directed Stars*. The whole algorithm presented in this chpater is divided into two sub-algorithms for an average

case, where the time complexity of the algorithm satisfies the assumption (4) as in Subsection 1.4.2 in Chapter 1.

The first one is concentrated on colouring and re-directing blue edges mechanism as summarised in Algorithm 7.1. The main task of the Algorithm 7.1 is to utilise blue edges that either are incident to vertices in a PDS or not; this, however, allows us to apply the proposed mechanism set forth in Subsection 7.4.2.1. Therefore, the most time-consuming part in the Algorithm 7.1 is to check blue edges in a given directed graph ($G^{'}$) such that there are at most $3^{E(b)}$ states. The second Algorithm 7.2 is based on *Rewiring Red Edges* between the *open neighbours* of $v \in PDS$, which is named in this chapter tails $t(n)$ and heads $h(n)$ of a *dependency path* (p). Therefore, there are at most $\big(h_n + |\{t_n - 1\}|\big)$ states. Consequently, the total running time of the whole Algorithms 7.1 and 7.2 is $\mathcal{O}(c^{E(b)} \cdot n)$ where there are at most $3^{E(b)} + \big(h_n + |\{t_n - 1\}|\big)$ states.

On the other hand, if there exist no sufficient blue edges that satisfy the cases in Definition 7.10, then the addition of red edges in a *Valid Colouring* as defined in Definition 7.8 should be applied for a worst case. This a lack of blue edges, however, can lead to increase the number of edges in a given directed graph and contradict with the main contribution of the chapter which is the study the effect of rewiring edges on *structural controllability* in order to achieve a minimal PDS while keeping the total number of edges unchanged.

7.6 Summary

This chapter proposed a reconstruction algorithm to attain a minimal PDS while keeping the total number of edges unchanged by applying rewiring edges which is based on DIRECTED STARS decomposition. This approach yields to the minimisation of the number of a PDS whilst still maintaining *structural controllability* through determining the number of *out-neighbours* of a PDS, and therefore, rewiring blue/red edges in terms of a *Valid Colouring* mechanism.

Chapter 8

Conclusions

8.1 Conclusion

Domination, a central topic in graph theory, becomes a relevant theme in the design and analysis of control systems, as it is an equivalent problem to that of *Kalman controllability*. There has been considerable interest in *structural controllability* originally introduced by Lin, which provides a graph-theoretical interpretation of Kalman controllability.

Structural controllability is a highly interesting concept for understanding vulnerabilities to attack in critical infrastructures, and recovery of (partial) controllability is often time-critical. Analysis of the *structural controllability* of the control graph over directed Erdős-Rényi graphs via the POWER DOMINATING SET problem was undertaken in this thesis; this provides an equal means of determining the control structure through identifying minimum *Driver Nodes*. The ability to identify *Driver Nodes* must be considered crucial for both attackers and defenders in control systems, as it is an obvious target for attackers wishing to disrupt the network control. We therefore study an alternative approach based on the POWER DOMINATING SET problem, which gives an equivalent formulation for identifying minimum *Driver Nodes* (N_D). This offers a strong motivation to study the ability of such systems to recover from deliberate attacks.

In Chapter 3, an overview of the problems of *controllability* and *structural controllability* as represented by the PDS problem was given. Also, an overview of the relevant literature was reviewed for different graph classes, whereby a PDS has been studied prior in order to identify a potential embedding of such structures

in Erdős-Rényi graphs for varied density and approximation characteristics, which may be realised for the purposes of making amendments to solve the DIRECTED POWER DOMINATING SET problem. This facilitated a speedy determination of feasible alternative control structures where adversaries have intercepted and corrupted the original control network as well as recovering of partial *controllability* should a control network become partitioned.

Chapter 4 provided a reconstruction algorithm for (directed) control graphs of bounded *tree-width* embedded in Erdős-Rényi random graphs based on recent works by Aazami and Stilp as well Guo *et al*. The algorithm takes account of the speedy redevelopment of a PDS facing threat as an elevated priority in light of PDS outcome optimisation, and therefore, purports an approximation inclusive of a dynamic programming approach for directed graphs, in which a tree of bounded width may be embedded within an Erdős-Rényi random graph.

In the following chapter, we also suggested a novel algorithm based a DFS structure, yielding an improved average-case complexity over previous work in Chapter 4, where an adversary with sufficient knowledge of the distribution of the network and the power domination relation can compromise controllability of dependent nodes or disconnect parts of the control original graph. This entails a minimising of the average-case complexity of the recovery algorithm through re-using of remaining fragments from the original control graph where permitted whilst identifying unutilised edges to minimise the number of a PDS.

Additionally, in Chapter 6, the *structural controllability* of the control graph in LTI via the PDS problem were studied. This addresses the question of how to recover a control graph as far as possible when the PDS or its dependent nodes are under adversarial attack without complete re-computation. Our method was sourced from a BLOCK DECOMPOSITION of a directed graph, which allows for the determination of both cut-vertices and cut-edges. This provides faster re-construction of a minimal PDS structure, and ultimately the re-gaining of control for operators of control systems through a three-step process.

Finally, In Chapter 7, we studied the instance of sparse Erdős-Rényi graphs with directed control edges, with a view to ascertaining the effect of rewiring edges on the *structural controllability* of directed Erdős-Rényi graphs so that a minimal PDS could be obtained while simultaneously preserving the number of edges unaffected. The approach lies in a DIRECTED STARS Decomposition of a directed graph, which allows us to determine the number of *out-neighbours* of a PDS, with a view to obtaining a minimal PDS.

8.2 Directions for Future Work

When it comes to the topic of constructing algorithms, specifically for the purpose of approximation *structural controllability* graphs via a Power Dominating Set, research efforts are needed to strengthen this area, particularly for the restoration of the structural controllability of different classes of graphs via the PDS problem when nodes are being attacke. There is a need for additional research if issues relating to the *structural controllability* of complex networks are to be solved and pave the way for the industry to adopt such approaches. Thus, our future investigations will focus on studying different classes of graphs and investigating cascading failure attack on network *controllability* as follows:

8.2.1 Studying Different Classes of Graphs

This thesis initially studied directed Erdős-Rényi random graphs, where they represent a widely studied class of graphs that has been extensively considered in respect of various problems concerning graph theory, and random graphs constitute an important and active research area, with numerous models that have been applied to communication networks.

However, further extensions will be conducted an investigation of how control structures of different classes of graphs as well as the approximation characteristics can be achieved to find solutions for directed graphs of different complex networks based on a PDS. This, however, includes applying the algorithms outlined in this thesis to adapt them for studying different classes of graphs.

8.2.2 Investigating Cascading Failure Attack on Network Controllability

As stated in this thesis, our previous research relied on an assumption where a given directed Erdős Renyi graph had been either randomly breakdown or intentionally attacks. However, we have not considered how a *controllability* graph had been partitioned or damaged. Thus, this further research shall focus on cascades provoked by the removal of the vertices and edges, and how susceptible different complex networks such as directed Erdős Renyi model of random networks are to a range of vertex and edge attacks. This also will include consideration of how *controllability* for networks evolves during cascading failures in the case of two distinct attack strategies, random and intentional.

Bibliography

[1] AAZAMI, A. Domination in Graphs with Bounded Propagation: Algorithms, Formulations and Hardness Results. *Journal of Combinatorial Optimization 19*, 4 (May 2012), 429–456. doi:10.1007/s10878-008-9176-7.

[2] AAZAMI, A., AND STILP, K. Approximation Algorithms and Hardness for Domination with Propagation. *SIAM Journal on Discrete Mathematics 23*, 3 (September 2009), 1382–1399. doi:10.1137/06066672X.

[3] AHANGAR, H. A., AND PUSHPALATHA, L. The Forcing Domination Number of Hamiltonian Cubic Graphs. *International Journal of Mathematical Combinatorics 2* (2009), 53–57.

[4] AHO, A. V., AND HOPCROFT, J. E. *Design and Analysis of Computer Algorithms*. Pearson Education India, 1974.

[5] AHUJA, M., AND ZHU, Y. An Efficient Distributed Algorithm for Finding Articulation Points, Bridges, and Biconnected Components in Asynchronous Networks. In *Foundations of Software Technology and Theoretical Computer Science* (1989), Springer-Verlag, pp. 99–108.

[6] ALBERT, R., JEONG, H., AND BARABÁSI, A.-L. Error and Attack Tolerance of Complex Networks. *Nature 406* (July 2000), 378–382. doi:10.1038/35019019.

[7] ALCARAZ, C., ETCHEVÉS MICIOLINO, E., AND WOLTHUSEN, S. D. Multi-Round Attacks on Structural Controllability Properties for Non-Complete Random Graphs. In *Proceedings of the 16th Information Security Conference*

(ISC 2013) (Dallas, TX, USA, Nov. 2013), Lecture Notes in Computer Science, Springer-Verlag.

[8] ALCARAZ, C., ETCHEVÉS MICIOLINO, E., AND WOLTHUSEN, S. D. Structural Controllability of Networks for Non-interactive Adversarial Vertex Removal. In *Proceedings of the 8th International Workshop on Critical Information Infrastructures Security (CRITIS 2013)* (Amsterdam, The Netherlands, Sept. 2013), vol. 8328 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 120–132. doi:10.1007/978-3-319-03964-0_11.

[9] ALCARAZ, C., AND WOLTHUSEN, S. *Recovery of Structural Controllability for Control Systems*. Springer Berlin Heidelber, Berlin, Heidelberg, 2014, pp. 47–63. doi:10.1007/978-3-662-45355-1_4.

[10] ALIMONTI, P., AND KANN, V. Hardness of Approximating Problems on Cubic Graphs. In *Algorithms and Complexity*, vol. 1203 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 1997, pp. 288–298. doi:10.1007/3-540-62592-5_80.

[11] ALT, H., BLUM, N., MEHLHORN, K., AND PAUL, M. Computing a Maximum Cardinality Matching in a Bipartite Graph in Time $O(n^{1.5}mlog\,n)$. *Information Processing Letters 37*, 4 (1991), 237–240. doi:10.1016/0020-0190(91)90195-N.

[12] ALWASEL, B., AND WOLTHUSEN, S. Structural Controllability Analysis via Embedding Power Dominating Set Approximation in Erdős-Rényi Graphs. In *the Proceedings of the 29th IEEE International Conference on Advanced Information Networking and Applications (AINA-2015)* (Gwangju, Korea, 2015), IEEE Press, pp. 418–423. doi:10.1109/WAINA.2015.77.

[13] ALWASEL, B., AND WOLTHUSEN, S. D. Reconstruction of Structural Controllability over Erdős-Rényi Graphs via Power Dominating Sets. In *Proceedings of the 9th Annual Cyber and Information Security Research Conference* (April 2014), CISR '14, ACM, pp. 57–60. doi:10.1145/2602087.2602095.

[14] ALWASEL, B., AND WOLTHUSEN, S. D. Recovering Structural Controllability on Erdős-Rényi Graphs via Partial Control Structure Re-Use. In *9th International Conference on Critical Information Infrastructures Security (CRITIS 2014)* (Limassol, Cyprus, October 2014), Springer-Verlag, pp. 293–307. doi:10.1007/978-3-319-31664-2_30.

[15] ALWASEL, B., AND WOLTHUSEN, S. D. Recovering Structural Controllability on Erdős-Rényi Graphs in the Presence of Compromised Nodes. In *10th International Conference on Critical Information Infrastructures Security (CRITIS 2015)* (Berlin, Germany, October 2015), Springer-Verlag, pp. 105–119. doi:10.1007/978-3-319-33331-1_9.

[16] ATKINS, D., HAYNES, T. W., AND HENNING, M. A. Placing Monitoring Devices in Electric Power Networks Modelled by Block Graphs. *Ars Combinatorica 79*, 1 (Apr. 2006).

[17] BAI, L., HOU, L., AND LAO, S. A Method for Enhancing Controllability with Adding Links in Directed Networks. In *Systems and Informatics (ICSAI), International Conference on* (May 2012), IEEE Press, pp. 13–15. doi:10.1109/ICSAI.2012.6223402.

[18] BALDWIN, T., MILI, L., BOISEN, M.B., J., AND ADAPA, R. Power System Observability with Minimal Phasor Measurement Placement. *Power Systems, IEEE Transactions on 8*, 2 (May 1993), 707–715. doi:10.1109/59.260810.

[19] BARRERA, R. On the Power Domination Problem in Graphs.

[20] BARRERA, R., AND FERRERO, D. Power Domination in Cylinders, Tori, and Generalized Petersen Graphs. *Networks 58*, 1 (2011), 43–49. doi:10.1002/net.20413.

[21] BARRETO, C., CÁRDENAS, A. A., AND QUIJANO, N. *Controllability of Dynamical Systems: Threat Models and Reactive Security*. Springer International Publishing, Cham, 2013, pp. 45–64. doi:10.1007/978-3-319-02786-9_4.

[22] BARTHÉLEMY, M. Betweenness Centrality in Large Complex Networks. *The European Physical Journal B: Condensed Matter and Complex Systems 38*, 2 (Mar. 2004), 163–168. doi:10.1140/epjb/e2004-00111-4.

[23] BELUR, M., AND CHAKRABORTY, D. Graph Theoretic Methods in the Study of Structural Issues in Control. In *ICCAS-SICE, 2009* (August 2009), pp. 3940–3944.

[24] BINKELE-RAIBLE, D., AND FERNAU, H. An Exact Exponential Time Algorithm for POWER DOMINATING SET. *Algorithmica 63*, 1–2 (June 2012), 323–346. doi:10.1007/s00453-011-9533-2.

[25] BOCCALETTI, S., LATORA, V., MORENO, Y., CHAVEZ, M., AND HWANG, D.-U. Complex Networks: Structure and Dynamics. *Physics Reports 424*, 4 (2006), 175–308. doi:10.1016/j.physrep.2005.10.009.

[26] BOLLOBÁS, B. *Random Graphs*, 2nd ed., vol. 73 of *Cambridge Studies in Advanced Mathematics*. Cambridge University Press, Cambridge, UK, 2001.

[27] BOLLOBÁS, B., AND RIORDAN, O. Robustness and Vulnerability of Scale-Free Random Graphs. *Internet Mathematics 1*, 1 (Jan. 2003), 1–35. doi:10.1080/15427951.2004.10129080.

[28] BRUENI, D. J. Minimal PMU Placement for Graph Observability: A Decomposition Approach.

[29] BRUENI, D. J., AND HEATH, L. S. The PMU Placement Problem. *SIAM Journal on Discrete Mathematics 19*, 3 (2005), 744–761. doi:10.1137/S0895480103432556.

[30] 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.

[31] COSTA, L. D. F., RODRIGUES, F. A., TRAVIESO, G., AND VILLAS BOAS, P. R. Characterization of Complex Networks: A Survey of Measure-

ments. *Advances in Physics 56*, 1 (May 2007), 167–242. doi:10.1080/00018730601170527.

[32] COURCELLE, B., MAKOWSKY, J. A., AND ROTICS, U. Linear Time Solvable Optimization Problems on Graphs of Bounded Clique-Width. *Theory of Computing Systems 33*, 2 (Apr. 2000), 125–150. doi:10.1007/s002249910009.

[33] COWAN, N. J., CHASTAIN, E. J., VILHENA, D. A., FREUDENBERG, J. S., AND BERGSTROM, C. T. Nodal Dynamics, Not Degree Distributions, Determine the Structural Controllability of Complex Networks. *Public Library of Science ONE 7*, 6 (June 2012), 1–5. doi:10.1371/journal.pone.0038398.

[34] DEAN, N., ILIC, A., RAMIREZ, I., SHEN, J., AND TIAN, K. On the Power Dominating Sets of Hypercubes. In *Computational Science and Engineering (CSE) IEEE 14th International Conference on* (August 2011), pp. 488–491. doi:10.1109/CSE.2011.89.

[35] DENNING, D. *Information Warfare and Security*. ACM Press Series. ACM Press, 1999.

[36] DIESTEL, R. *Graph Theory*, 2nd edition ed. Electronic Library of Mathematics. Springer New York, 2000.

[37] DORBEC, P., MOLLARD, M., KLAVŽAR, S., AND ŠPACAPAN, S. Power Domination in Product Graphs. *SIAM Journal on Discrete Mathematics 22*, 2 (2008), 554–567. doi:10.1137/060661879.

[38] DORFLING, M., AND HENNING, M. A. A Note on Power Domination in Grid Graphs. *Discrete Applied Mathematics 154*, 6 (Apr. 2006), 1023–1027. doi:10.1016/j.dam.2005.08.006.

[39] DOWNEY, R. G., AND FELLOWS, M. R. *Parameterized Complexity*. Monographs in Computer Science. Springer-Verlag, Heidelberg, Germany, 1999.

[40] ERDŐS, P., AND RÉNYI, A. On Random Graphs. I. *Publ. Math. Debrecen 6* (1959), 290–297.

[41] FARWELL, J. P., AND ROHOZINSKI, R. Stuxnet and the Future of Cyber War. *Survival 53*, 1 (2011), 23–40. doi:10.1080/00396338.2011.555586.

[42] FEIGE, U. A Threshold of $\ln n$ for Approximating Set Cover. *Journal of the ACM 45*, 4 (July 1998), 634–652. doi:10.1145/285055.285059.

[43] FLUM, J., AND GROHE, M. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. Springer-Verlag, Berlin, Germany, 2006.

[44] FOSTER, J. G., FOSTER, D. V., GRASSBERGER, P., AND PACZUSKI, M. Edge Direction and the Structure of Networks. *Proceedings of the National Academy of Sciences 107*, 24 (Jan. 2010), 10815–10820. doi:10.1073/pnas.0912671107.

[45] GALLOS, L. K., COHEN, R., LILJEROS, F., ARGYRAKIS, P., BUNDE, A., AND HAVLIN, S. Attack Strategies on Complex Networks. In *Proceedings of the 6th International Conference on Computational Science (ICCS 2006 Part III)* (Reading, UK, May 2006), vol. 3993 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 1048–1055. doi:10.1007/11758532_143.

[46] GIBBONS, A. *Algorithmic Graph Theory*. Cambridge University Press, 1985.

[47] GOLOMB, S. W., AND BAUMERT, L. D. Backtrack Programming. *J. ACM 12*, 4 (Oct. 1965), 516–524. doi:10.1145/321296.321300.

[48] GUO, J., HÜFFNER, F., AND NIEDERMEIER, R. A Structural View on Parameterizing Problems: Distance from Triviality. In *Proceedings of the First International Workshop on Parameterized and Exact Computation (IWPEC 2004)* (Bergen, Norway, Sept. 2004), vol. 3162 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 162–173. doi:10.1007/978-3-540-28639-4_15.

[49] GUO, J., NIEDERMEIER, R., AND RAIBLE, D. Improved Algorithms and Complexity Results for Power Domination in Graphs. *Algorithmica 52*, 2 (Oct. 2008), 177–202. doi:10.1007/s00453-007-9147-x.

[50] HARANT, J., PRUCHNEWSKI, A., AND VOIGT, M. On Dominating Sets and Independent Sets of Graphs. *Combinatorics, Probability and Computing 8*, 06 (1999), 547–553.

[51] HARARY, F. *Graph Theory*. Addison-Wesley Series in Mathematics. Perseus Books, 1994.

[52] HAYNES, T. W., HEDETNIEMI, S. M., HEDETNIEMI, S. T., AND HENNING, M. A. Domination in Graphs Applied to Electric Power Networks. *SIAM Journal on Discrete Mathematics 15*, 4 (Aug. 2002), 519–529. doi:10.1137/S0895480100375831.

[53] HOLME, P., KIM, B. J., YOON, C. N., AND HAN, S. K. Attack Vulnerability of Complex Networks. *Physical Review E 65*, 5 (May 2002), 056109. doi:10.1103/PhysRevE.65.056109.

[54] HON, W.-K., LIU, C.-S., PENG, S.-L., AND TANG, C. Y. Power Domination on Block-Cactus Graphs. In *The 24th Workshop on Combinatorial Mathematics and Computation Theory* (2007), pp. 280–284.

[55] HOPCROFT, J. E., AND KARP, R. M. A $n^{5/2}$ Algorithm for Maximum Matchings in Bipartite Graphs. In *Switching and Automata Theory, 12th Annual Symposium on* (October 1971), IEEE Press, pp. 122–125. doi:10.1109/SWAT.1971.1.

[56] ITALIANO, G. Strong Bridges and Strong Articulation Points of Directed Graphs. In *SOFSEM 2012: Theory and Practice of Computer Science* (2012), vol. 7147 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, pp. 43–43. doi:10.1007/978-3-642-27660-6_4.

[57] JARCZYK, J., SVARICEK, F., AND ALT, B. Strong Structural Controllability of Linear Systems Revisited. In *Decision and Control and European Control Conference (CDC-ECC)* (December 2011), IEEE Press, pp. 1213–1218. doi:10.1109/CDC.2011.6160392.

[58] JI, M., AND EGERSTEDT, M. Distributed Coordination Control of Multiagent Systems while Preserving Connectedness. *Robotics, IEEE Transactions on 23*, 4 (August 2007), 693–703. doi:10.1109/TRO.2007.900638.

[59] KALMAN, R. E. Mathematical Description of Linear Dynamical Systems. *Journal of the Society of Industrial and Applied Mathematics Control Series A 1* (1963), 152–192.

[60] KAO, K., CHANG, J.-M., WANG, Y., XU, S.-H., AND JUAN, J. S.-T. Power Domination in Honeycomb Meshes. *Journal of Information Science and Engineering 29*, 6 (2013), 1249–1263.

[61] KIKUNO, T., YOSHIDA, N., AND KAKUDA, Y. A Linear Algorithm for the Domination Number of a Series-Parallel Graph. *Discrete Applied Mathematics 5*, 3 (1983), 299 – 311. doi:10.1016/0166-218X(83)90003-3.

[62] KLOKS, T. *Treewidth: Computations and Approximations*, vol. 842 of *Lecture Notes in Computer Science*. Springer-Verlag, Heidelberg, Germany, 1994.

[63] KNEIS, J., MÖLLE, D., RICHTER, S., AND ROSSMANITH, P. Parameterized Power Domination Complexity. *Information Processing Letters 98*, 4 (May 2006), 145–149. doi:10.1016/j.ipl.2006.01.007.

[64] KUHN, H. W. The Hungarian Method for the Assignment Problem. *Naval Research Logistics Quarterly 2*, 1-2 (1955), 83–97.

[65] LAI, Y.-L., CHIEN, P.-K., CHOU, S.-C., AND KAO, Y.-K. On Power Domination of Generalized Petersen Graphs. In *The 29th Workshop on Combinatorial Mathematics and Computation Theory* (2012), pp. 202–207.

[66] LANG, S. *Linear Algebra*. Springer Undergraduate Texts in Mathematics and Technology. Springer, 1987.

[67] LIAO, C.-S., AND LEE, D. Power Domination in Circular-Arc Graphs. *Algorithmica 65*, 2 (2013), 443–466. doi:10.1007/s00453-011-9599-x.

[68] LIAO, C.-S., AND LEE, D.-T. Power Domination Problem in Graphs. In *Proceedings of the 11th Annual International Conference on Computing and Com-*

binatorics (COCOON 2005) (Kunming, China, Aug. 2005), vol. 3595 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 818–828. doi:10.1007/11533719_83.

[69] LIN, C.-T. Structural Controllability. *IEEE Transactions on Automatic Control 19*, 3 (June 1974), 201–208. doi:10.1109/TAC.1974.1100557.

[70] LIOTTA, G., TAMASSIA, R., AND TOLLIS, I. *Graph Algorithms and Applications 4*. No. 4. World Scientific, 2006.

[71] LIU, Y.-Y., SLOTINE, J.-J., AND BARABÁSI, A.-L. Controllability of Complex Networks. *Nature 473* (May 2011), 167–173. doi:10.1038/nature10011.

[72] LIU, Y.-Y., SLOTINE, J.-J., AND BARABÁSI, A.-L. Control Centrality and Hierarchical Structure in Complex Networks. *Public Library of Science ONE 7*, 9 (Sept. 2012), 1–7. doi:10.1371/journal.pone.0044459.

[73] LOMBARDI, A., AND HÖRNQUIST, M. Controllability Analysis of Networks. *Physical Review E 75*, 5 (May 2007), 056110. doi:10.1103/PhysRevE.75.056110.

[74] LOVÁSZ, L., AND PLUMMER, M. D. *Matching Theory*. American Mathematical Society, Providence, RI, USA, 2009.

[75] LV-LIN, H., SONG, L., GANG, L., AND LIANG, B. Controllability and Directionality in Complex Networks. *Chinese Physics Letters 29*, 10 (2012), 108901. doi:10.1088/0256-307X/29/10/108901.

[76] LVLIN, H., SONGYANG, L., JIANG, B., AND LIANG, B. Enhancing Complex Network Controllability by Rewiring Links. In *Intelligent System Design and Engineering Applications (ISDEA), Third International Conference on* (January 2013), pp. 709–711. doi:10.1109/ISDEA.2012.168.

[77] MELCHIONNA, A., CALOCA, J., SQUIRES, S., ANTONSEN, T. M., OTT, E., AND GIRVAN, M. Impact of Imperfect Information on Network Attack. *Phys. Rev. E 91* (March 2015), 032807. doi:10.1103/PhysRevE.91.032807.

[78] MESBAHI, M., AND EGERSTEDT, M. *Graph Theoretic Methods in Multiagent Networks*. Princeton University Press, 2010.

[79] MICALI, S., AND VAZIRANI, V. V. An $O(\sqrt{|V|}|E|)$ Algorithm for Finding Maximum Matching in General Graphs. In *Proceedings of the 21st Annual Symposium on Foundations of Computer Science (FOCS 1980)* (Syracuse, NY, USA, Oct. 1980), IEEE Press, pp. 17–27. doi:10.1109/SFCS.1980.12.

[80] MILI, L., BALDWIN, T., AND PHADKE, A. Phasor Measurements for Voltage and Transient Stability Monitoring and Control. In *Workshop on Application of Advanced Mathematics to Power Systems* (San Francisco, September 1991), pp. 4–6.

[81] MOTTER, A. E., AND LAI, Y.-C. Cascade-Based Attacks on Complex Networks. *Physical Review E – Statistical, Nonlinear, and Soft Matter Physics 66*, 6 (July 2002), 378–382. doi:10.1103/PhysRevE.66.065102.

[82] NACHER, J. C., AND AKUTSU, T. Structural Controllability of Unidirectional Bipartite Networks. *Nature Scientific Reports 3*, 1647 (Apr. 2013), 1–7. doi:10.1038/srep01647.

[83] NEPUSZ, T., AND VICSEK, T. Controlling Edge Dynamics in Complex Networks. *Nature Physics 8*, 7 (July 2012), 568–573. doi:10.1038/nphys2327.

[84] NEWMAN, M. E. The Structure and Function of Complex Networks. *SIAM review 45*, 2 (2003), 167–256.

[85] NIE, S., WANG, X., ZHANG, H., LI, Q., AND WANG, B. Robustness of Controllability for Networks Based on Edge-Attack. *Public Library of Science ONE 9*, 2 (Feb. 2014), 1–8. doi:10.1371/journal.pone.0089066.

[86] OGATA, K. *Modern Control Engineering*, 4th ed. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2001.

[87] PAI, K.-J., CHANG, J.-M., AND WANG, Y.-L. A Simple Algorithm for Solving the Power Domination Problem on Grid Graphs. In *Proc. the 24th Workshop Combin. Math. and Comput. Theory* (2007), pp. 256–260.

[88] PAI, K.-J., CHANG, J.-M., AND WANG, Y.-L. Restricted Power Domination and Fault-Tolerant Power Domination on Grids. *Discrete Applied Mathematics 158*, 10 (2010), 1079–1089. doi:10.1016/j.dam.2010.03.001.

[89] PÓSFAI, M., LIU, Y.-Y., SLOTINE, J.-J., AND BARABÁSI, A.-L. Effect of Correlations on Network Controllability. *Nature Scientific Reports 3*, 1067 (Jan. 2013), 1–7. doi:10.1038/srep01067.

[90] POWER, R. *Tangled Web: Tales of Digital Crime from the Shadows of Cyberspace*. Macmillan Press Ltd, Basingstoke, UK, 2000.

[91] PU, C.-L., PEI, W.-J., AND MICHAELSON, A. Robustness Analysis of Network Controllability. *Physica A: Statistical Mechanics and its Applications 391*, 18 (September 2012), 4420–4425. doi:10.1016/j.physa.2012.04.019.

[92] PUNTAMBEKAR, A. *Advanced Data Structures and Algorithms*. Technical Publications, 2008.

[93] ROBERTSON, N., AND SEYMOUR, P. Graph Minors. II. Algorithmic Aspects of Tree-Width. *Journal of Algorithms 7*, 3 (1986), 309–322. doi:10.1016/0196-6774(86)90023-4.

[94] RUTHS, J., AND RUTHS, D. Robustness of Network Controllability under Edge Removal. In *Complex Networks IV*, vol. 476 of *Studies in Computational Intelligence*. Springer Berlin Heidelberg, 2013, pp. 185–193. doi:10.1007/978-3-642-36844-8_18.

[95] SCHNEIDER, C. M., MOREIRA, A. A., ANDRADE, JR., J. S., HAVLIN, S., AND HERRMANN, H. J. Mitigation of Malicious Attacks on Networks. *Proceedings of the National Academy of Sciences of the United States of America 108*, 10 (Mar. 2011), 3838–3841. doi:10.1073/pnas.1009440108.

[96] SHIELDS, R., AND PEARSON, J. Structural Controliability of Multi-Input Linear Systems. In *Decision and Control including the 14th Symposium on Adaptive Processes, IEEE Conference on* (December 1975), pp. 807–809. doi:10.1109/CDC.1975.270615.

[97] SLOTINE, J.-J., AND LIU, Y.-Y. Complex Networks: The Missing Link. *Nat Phys 8*, 7 (2012), 512–513. doi:10.1038/nphys2342.

[98] SUDAKOV, B., AND VU, V. H. Local Resilience of Graphs. *Random Structures and Algorithms 33*, 4 (Aug. 2008), 409–433. doi:10.1002/rsa.20235.

[99] TANNER, H. On the Controllability of Nearest Neighbor Interconnections. In *Decision and Control. CDC. 43rd IEEE Conference on* (December 2004), vol. 3, pp. 2467–2472. doi:10.1109/CDC.2004.1428782.

[100] TARJAN, R. Depth-First Search and Linear Graph Algorithms. *SIAM Journal on Computing 1*, 2 (1972), 146–160.

[101] TEIXEIRA, A., PÉREZ, D., SANDBERG, H., AND JOHANSSON, K. H. Attack Models and Scenarios for Networked Control Systems. In *Proceedings of the 1st International Conference on High Confidence Networked Systems* (New York, NY, USA, 2012), HiCoNS '12, ACM, pp. 55–64. doi:10.1145/2185505.2185515.

[102] TERRELL, W. J. Some Fundamental Control Theory I: Controllability, Observability, and Duality. *The American Mathematical Monthly 106*, 8 (1999), 705–719.

[103] VARGHESE, S., AND VIJAYAKUMAR, A. Power Domination in Some Classes of Graphs. *EuroComb'11* (2011).

[104] WANG, B., GAO, L., GAO, Y., AND DENG, Y. Maintain the Structural Controllability under Malicious Attacks on Directed Networks. *Europhysics Letters 101*, 5 (Mar. 2013), 1–6. doi:10.1209/0295-5075/101/58003.

[105] WANG, W.-X., NI, X., LAI, Y.-C., AND GREBOGI, C. Optimizing Controllability of Complex Networks by Minimum Structural Perturbations. *Physical Review E 85*, 2 (Feb. 2012), 026115. doi:10.1103/PhysRevE.85.026115.

[106] XU, G., AND KANG, L. On the Power Domination Number of the Generalized Petersen Graphs. *Journal of Combinatorial Optimization 22*, 2 (2011), 282–291. doi:10.1007/s10878-010-9293-y.

[107] XU, G., KANG, L., SHAN, E., AND ZHAO, M. Power Domination in Block Graphs. *Theoretical Computer Science 1-3* (Aug. 2006), 299–305. doi:10.1016/j.tcs.2006.04.011.

[108] ZABCZYK, J. *Mathematical Control Theory: An Introduction*. Modern Birkhäuser Classics. Birkhäuser Boston, 2009.

[109] ZHAO, M., KANG, L., AND CHANG, G. J. Power Domination in Graphs. *Discrete Mathematics 15*, 6 (Aug. 2006), 1812–1816. doi:10.1016/j.disc.2006.03.037.