The Distance-based Critical Node Detection Problem: Models and Algorithms

Glory Uche Alozie

Department Of Management Science

University of Strathclyde

Glasgow, UK

2021

A thesis submitted in fulfilment of the requirement for the degree of Doctor of Philosophy

Declaration

This thesis is the result of the author's original research. It has been composed by the author and has not been previously submitted for examination which has led to the award of a degree.

Signed: Glory Uche Alozie Date: June 4, 2021

Acknowledgements

I would like to express my gratitude to my amazing supervisors Prof Kerem Akartunalı, and Dr Ashwin Arulselvan under whose guidance I have become better at research. Thank you for your availability, encouragement and unwavering support not just towards completing my PhD in time but also towards my professional development and post-doctoral career.

I am grateful to the University of Strathclyde and the Air Force Office of Scientific Research for funding the research studentship through which I was able to complete my PhD research.

I would like to thank Alison and Elaine (both of them) at the department for providing excellent service as it concerns administrative matters of the PhD.

To my parents and siblings, thank you for your prayers and care. My spiritual family in Glasgow, the Redeemed Christian Church of God, Beautiful Gate, thanks for the different ways you made my student journey in Glasgow a rewarding one.

To friends and colleagues at Strathclyde Business School, having open conversations with you all helped me know that I was not alone in the journey. Thanks to my colleagues (past and present) at Management Science for your support and concern during my maternity.

A very special appreciation to my beloved husband for the sacrifices you made to ensure that I finished my research. Thanks for your prayers, encouragement and hard work to take care of Mercy and I, you are the best. To my daughter (my personal assistant), having you during my PhD and watching you grow has been a joyful experience. Doing PhD with you has made me more resilient, you are a blessing.

To my heavenly Father, You are the reason I am here. Thank You for remaining faithful to Your word and furnishing me with grace to do all I need to do.

Publications

Parts of this dissertation have been published or accepted for publication in peer-reviewed journals, as listed below:

- Alozie, G.U., Arulselvan, A., Akartunalı, K., Pasiliao Jr, E.L (2021) Efficient methods for the distance-based critical node detection problem in complex networks. *Computers & Operations Research*, 131 (1):105254. DOI: 10.1016/j.cor.2021.105254.
- Alozie, G.U., Arulselvan, A., Akartunalı, K., Pasiliao Jr, E.L (2021) A heuristic approach for the distance-based critical node detection problem in complex networks. *Journal of the Operational Research Society.* DOI: 10.1080/01605682.2021.1913078

Dedication

To my husband and daughter...

Victor & Mercy

Abstract

In the wake of terrorism and natural disasters, assessing networked systems for vulnerability to failures that arise from these events is essential to maintaining the operations of the systems. This is very crucial given the heavy dependence of daily social and economic activities on networked systems such as transport, telecommunication and energy networks as well as the interdependence of these networks. In this thesis, we explore methods to assess the vulnerability of networked systems to element failures which employ connectivity as the performance measure for vulnerability. The associated optimisation problem termed the critical node (edge) detection problem seeks to identify a subset of nodes (edges) of a network whose deletion (failure) optimises a network connectivity objective. Traditional connectivity measures employed in most studies of the critical node detection problem overlook internal cohesiveness of networks and the extent of connectivity in the network. This limits the effectiveness of the developed methods in uncovering vulnerability with regards to network connectivity. Our work therefore focuses on distance-based connectivity which is a fairly new class of connectivity introduced for studying the critical node detection problem to overcome the limitations of traditional connectivity measures.

In Chapter 1, we provide an introduction outlining the motivations and the methods related to our study. In Chapter 2, we review the literature on the critical node detection problem as well as its application areas and related problems. Following this, we formally introduce the distance-based critical node detection problem in Chapter 3 where we propose new integer programming models for the case of hop-based distances

and an efficient algorithm for the separation problems associated with the models. We also propose two families of valid inequalities. In Chapter 4, we consider the distance-based critical node detection problem using a heuristic approach in which we propose a centrality-based heuristic that employs a backbone crossover and a centrality-based neighbourhood search. In Chapter 5, we present generalisations of the methods proposed in Chapter 3 to edge-weighted graphs. We also introduce the edge-deletion version of the problem which we term the distance-based critical edge detection problem. Throughout Chapters 3, 4 and 5, we provide computational experiments.

Finally, in Chapter 6 we present conclusions as well future research directions.

Keywords: Network Vulnerability, Critical Node Detection Problem, Distance-based Connectivity, Integer Programming, Lazy Constraints, Branch-and-cut, Heuristics

Abbreviations

3C-CNP	Component Cardinality Constrained Critical Node Problem
BIP	Binary Integer Programming
BFS	Breadth First Search
CC-CNP	Cardinality Constrained Critical Node Problem
CG	Chvátal-Gomory
CNDP	Critical Node Detection Problem
CNP	Critical Node Problem
DCEDP	Distance-based Critical Edge Detection Problem
DCNDP	Distance-based Critical Node Detection Problem
DCNDP-1	Distance-based Critical Node Detection Problem class 1 distance function
DCNDP-2	Distance-based Critical Node Detection Problem class 2 distance function
DCNDP-3	Distance-based Critical Node Detection Problem class 3 distance function
DCNDP-base	Distance-based Critical Node Detection Problem base model

DCNDP-PBM	Distance-based Critical Node Detection Problem aggregated Path Based Model
DCNDP-PBML	Distance-based Critical Node Detection Problem disaggregated Path Based Model
DFS	Depth First Search
ECM	Enhanced Compact Model
GRASP	Greedy Randomised Adaptive Search Procedure
ILP	Integer Linear Programming
IP	Integer Programming
KPP	Key Player Problem
LB	Lower Bound
LP	Linear Programming
MILP	Mixed Integer Linear Programming
MIP	Mixed Integer Programming
NDPVC	Network Design Problem with Vulnerability Constraints
PBIL	Population Based Incremental Learning
PBM	Path Based Model
PR	Path Relinking
SNA	Social Network Analysis
UB	Upper Bound
VNS	Variable Neighbourhood Search

Notations and Symbols

$\mathbb{R}_+^n$	The n-dimensional space of non-negative real numbers
$\mathbb{R}_+$	The set of non-negative real numbers
$\mathbb{Z}_+^n$	The n-dimensional space of non-negative integer numbers
$conv(X)$	Convex hull of X
$O()$	Big-O notation indicating problem complexity
$\mathcal{NP}$	Complexity class
$Proj_{\mathcal{X}}(P)$	Projection of P onto $\mathcal{X}$
$\forall$	Equation/inequality applies for all indices in the indicated set
$\exists$	Element exists in the given set
$V(G)$	Set of vertices/nodes in graph G
$E(G)$	Set of edges/arcs in graph G
G^R	The resultant subgraph of G obtained by deleting subset of nodes R
$\mathcal{N}_G(i)$	The set of nodes adjacent to node i in graph G
$d_G(i,j)$	The shortest path length between nodes i and j in graph G
$\mathcal{P}_L(i,j)$	The set of paths of length at most L connecting nodes i and j

List of Figures

List of Tables

Contents

Chapter 1

Introduction

Many real life systems and dynamics occurring within them are often represented as networks. Simply put, a network is a "collection of objects in which some pairs of these objects are connected by links" (Easley & Kleinberg 2010, p 2). These objects often referred to as nodes (or vertices) could be any entity such as human beings, cities, airports, computer servers, power generators etc while the links (or edges) reflect relationships, communication and transmission that occur between them. The ability to represent systems as networks provides tools for understudying natural systems with the aim of addressing interesting issues that arise in practice. One of such issues entails identification of parts of a system that are critical to the maintenance of performance of the system. This is crucial because many functions or operations that occur within a network are influenced by a fraction of its nodes (or edges) whose failure would cause a significance degradation of network performance. For example, in a social network context, the speed of diffusion of information across members of a given social network is largely dependent on a few "influential" individuals. Hence, for a case of malicious information in a social network, the spread could be curtailed by identifying those influential individuals for monitoring or temporal removal from the network.

Furthermore, economic and social infrastructures such as energy, health, transportation, telecommunication etc are largely interdependent. The implication of this is that a major disruption in one of these infrastructures would have a cascading effect on other

infrastructures which further impacts the entire economy. For instance, natural disasters such as flood and hurricane not only disrupt transport systems such as rail and road network but sometimes lead to power outages which further impact telecommunication networks. Given the uncertainty of these events as well as the widespread disruptions they have on social and economic activities, disaster management planners are becoming more strategic and proactive. One important strategy is the assessment of network infrastructures for potential vulnerabilities. This primarily involves identifying important elements (nodes and/or arcs) of a network whose loss would lead to the worst degradation of the network performance so as to fortify them. Depending on the definition of network performance, different notions of node importance have appeared in literature. Popular notions common in literature include: most vital nodes (Corley & David 1982), most influential nodes (Kempe et al. 2005), key players (Borgatti 2006) and critical nodes (Arulselvan et al. 2009). This research focuses on identification of critical nodes which are nodes that are important in ensuring connectivity in a network. The associated optimisation problem is termed the critical node detection problem (CNDP) formally defined as follows:

> *Given a network* $G = (V, E)$ *with* $n = |V|$ *nodes and* $m = |E|$*edges, identify a subset of nodes of a specified cardinality* B *whose deletion achieves optimum objective with respect to predefined connectivity measure*

A major aspect of the critical node detection problem lies in the definition of appropriate connectivity measure which is largely influenced by the application objective being considered. Thus, a node might be part of a critical node set for one application problem but in another problem it is not critical. Network connectivity metrics in foremost studies on the critical node detection problem have an underlying objective of fragmentation whereby disconnection is restricted to the absence of a path between pairs of node in a network. This definition of disconnectivity is limiting since in many application contexts, if the path(s) connecting nodes in a network have a sufficiently large length, then such nodes could be seen as practically disconnected. For instance, in a transportation network problem, while a source-destination route might exist, if

the duration of a trip via the available route precludes attainment of a time-bound activity, then such is a practical disconnection. Another interesting application is in biological networks where the length of a path between two nodes is analogous to the number of steps it takes to complete a biochemical process. Thus, a chemical reaction cannot effectively take place if the biochemical process is slow enough (Aringhieri et al. 2019). Also, in communication and telecommunication network, speed and coverage are key operational issues for assessing connectivity and both of these issues are related to distance between nodes in the network.

In this research, we focus on distance-based connectivity metrics in the study of the critical node detection problem. We present motivations for the research problem and introduce important methods that inform the methodological framework of our research. We then formalise our research question and our major research contributions. Finally, we conclude the chapter with an outline of the thesis.

1.1 Motivation

Networks are ubiquitous in real world. Transportation networks (rail, air, and road) provide a means of commuting people and goods from one location to another.

Telecommunication networks allow seamless communication between individuals and groups of individuals located across different continents as well as access to financial transactions. Energy networks give us daily access to energy supply. The volume of people and information exchange on different social networks such as facebook, twitter etc is an indication of the quest for connection. With the capacity for reachability that network affords us comes the price of vulnerability to attacks. While networks are built and maintained to establish ease in connection, unscrupulous agents are exploring the world's "connectedness" to spread malicious items through these networks leading to system breakdown and security compromise. Besides targeted attacks by enemies, natural occurrences like environment hazards also pose a threat to economic infrastructures.

Network designers are beginning to consider the overarching issue of survivability which has given rise to research efforts in the area of survivable network design (see discussions and references in Gouveia & Leitner (2017) and Gouveia et al. (2018)). For existing networks, vulnerability assessment becomes a pro-active step towards curtailing the impact of attacks and failures on the overall network performance. A very useful strategy for assessing vulnerability in networks is to identify elements of a network whose deletion (caused by failure or attacks) would compromise network connectivity. These network elements are termed critical nodes and edges. When the elements being considered are nodes, the associated optimisation problem is the so-called critical node detection problem (Arulselvan et al. 2009).

The growing interest on node and edge deletion problems in networks and in particular the critical node detection problem (CNDP) is motivated by its application to varieties of real life problems springing from different fields. In disease epidemiology where an outbreak of infectious disease must be contained, the CNDP find useful application. Interventions would either be vaccination of susceptible persons or removal of infected persons to contain the spread of the disease. In both cases, we are constrained by the number of persons to vaccinate or infected persons to remove owing to cost or availability of vaccines and ethical issues surrounding isolating individuals. Thus, a realistic strategy would be to identify members of the population to vaccinate or isolate in order to minimise the spread. Specific optimisation objectives in relation to this include minimisation of the number of infected-susceptible node connections and minimisation of the total number of susceptible nodes connected to one or more infected nodes (Nandi & Medal 2016). This is also similar in telecommunication networks, where critical nodes (servers) need to be disconnected to stop the spread of a virus (malware) through the network. In the study of covert terrorist network, key individuals of the terrorist network would be targeted for "neutralisation" so as to dismantle communication between groups or individuals in the network (Arulselvan et al. 2009). The CNDP has also been identified as potentially useful for drug design applications in the study of protein-protein interactions networks (Boginski & Commander 2009). In transportation network context,

edge deletion has also been applied. For instance, given some source-destination points $(s-t)$, road segments or highways whose disruption would maximally compromise traffic flow from source to destination represent critical edges (Matisziw & Murray 2009).
Inspired by these application problems, researchers in mathematical optimization have developed mathematical models to study the critical node detection problem with the underlying objective being to optimise some connectivity measure. Common objectives of traditional models for the critical node detection problem include minimisation of the total number of connected node pairs in the resultant subgraph (Arulselvan et al. 2009), maximisation of the total number of connected components in the residual graph (Shen & Smith 2012) and minimisation of the size of the largest connected component in the residual graph (Shen et al. 2012). Although these objectives represent relevant real life applications, the associated optimisation models are not very effective in uncovering vulnerability in some other applications. In the rest of the section, we present some limitations of these fragmentation CNDP objective that constitute the motivations for our research.

Distance-based connectivity

Traditional models for critical node detection problem have the underlying goal of network fragmentation. Thus popular objectives such as minimisation of number of connected node pairs and maximisation of number of connected components in the residual graph are only concerned with whether or not there exist a path between node pairs. However, for some real-world applications such as in social and communication networks, "nodes do not have to be truly disconnected in order to be practically disconnected - if distances are long enough, the nodes can be seen as effectively separated" (Borgatti 2006). The practical influence of distance-based connectivity is evident in our daily or weekly choice of a mode of transport and service provider. For example, recall the last time you had to book a flight to a new unfamiliar destination t or for a business trip. Although there were several options to your destination, some of them were undesirable due to number of stops or duration (including layovers). Thus, those options

that involved long haul flights with many stops, long wait times at the connecting airports and hence late night arrivals at an unfamiliar destination could be seen as practical disconnection. Another important distance-based connectivity objective is diffusion speed or communication efficiency. Communication efficiency is assumed to be inversely proportional to distances between nodes. Thus, individuals who are directly connected to information sources are more likely to have quicker and more reliable access to information than those with many intermediaries in between. In telecommunication

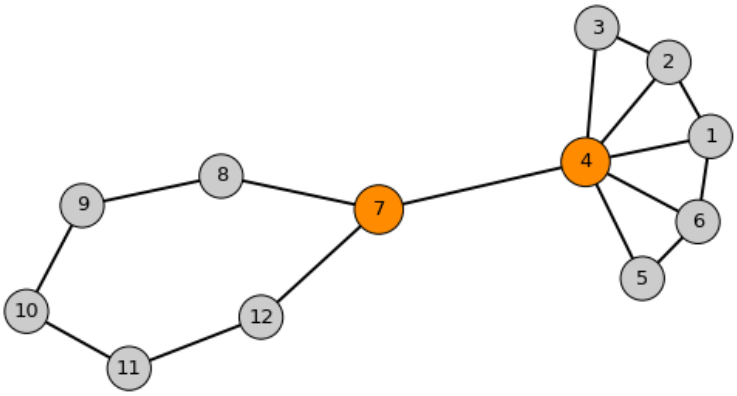


Figure 1.1: Graphical illustration of where a single node deletion based on fragmentation objectives could lead to sub-optimal solution for a distance-based application (Adapted from Veremyev, Prokopyev & Pasiliao (2014))

network design problems, quality of service (QoS) is very crucial for service providers and hop distance (number of edges) of a path is used as a measure of the level of service (such as bandwidth, delay etc) specified for each commodity (see e.g Balakrishnan & Altinkemer (1992), Guérin & Orda (2002), Chemodanov et al. (2018)). For an empirical illustration of the relevance of distance-based connectivity metrics, consider the graph in Figure 1.1 in which a maximum of one node is to be deleted. Based on the goal of network fragmentation, deletion of either node 4 or 7 achieves equal value on all

fragmentation-based objectives. In particular, deletion of either nodes results in a subgraph consisting of 2 connected components, 25 connected node pairs and largest component size of 6. Consider however, a case where service levels requirement for specified commodities is hop distance at most 3. It is easy to see that deletion of node 4 compromises this quality of service for commodity $(3, 5)$ whereas deletion of node 7 does not. Similarly with respect to commodity $(8, 12)$, node 7 is critical whereas node 4 is not. Hence, in such applications, use of fragmentation-based metrics would result in sub-optimal solutions which could have grave consequence for the decision maker.

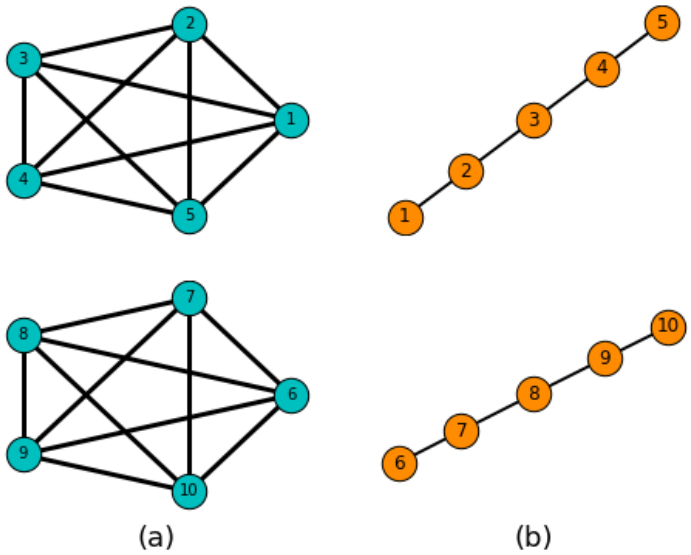


Figure 1.2: Graphical illustration of where a node deletion based on fragmentation does not take into account internal structure/distances (Adapted from Borgatti (2006))

The importance of distance-based metrics in network vulnerability assessment has been reported in literature. In a study on network robustness, Schieber et al. (2016), concluded through empirical analyses that distance-based metrics were more consistent in capturing structural deviations from the input network suggesting the importance of distance-based connectivity. Similarly, Borgatti (2006) observed that fragmentation-based

objectives overlook the internal structure of components formed. This is illustrated in Figure 1.2. Fragmentation-based metrics would indeed see the network in Figure 1.2(a) as equally fragmented as the network in Figure 1.2(b). However, one can see that distances (representing transmission lag or transportation time) are much higher in the latter network whose connected components are line graphs.

Infeasibility and budgetary limitations

Due to cohesive topological structures of certain input network, it is not possible to totally fragment a network by removal of a few nodes. Sometimes, a large proportion of nodes would need to be deleted to achieve the goal of fragmentation which might be "expensive" in practice. It has been argued that the topological structure of input networks and budgetary restriction on critical node set impact the possibility as well as the ease of solving associated instances of the CNDP (Veremyev, Prokopyev & Pasiliao 2014). So, when it is infeasible or rather expensive to achieve the goal of fragmentation, an alternative means of identifying nodes that are critical to maintaining network connectivity becomes imperative. The distance-based critical node detection problem provides a range of alternative metrics for evaluating what nodes are more critical than others without an explicit requirement of network fragmentation.

The aforementioned limitations of traditional fragmentation-based connectivity metrics for the critical node detection problem provide some motivations for this research project. The scope of the research is on the distance-based critical node detection problem (DCNDP) which was first introduced by Veremyev et al. (2015). In a nutshell, the DCNDP seeks a subset of nodes of specified cardinality whose deletion optimises some distance-based connectivity metric in the residual graph. Unlike the traditional CNDP, the distance-based critical node detection problem does not just consider whether or not nodes are connected but accounts for the extent or cost of connections. In comparison to the traditional fragmentation-based critical node detection problem, research on the distance-based critical node detection problem is quite limited. This is partly due to the

the fact that the distance-based CNDP has only been in existence for five years with only two computational studies which further motivates the direction of this research. Our methodological approach to studying the DCNDP is motivated by the computational limitation of the existing model as well as the need for extensive empirical analyses to support decision makers in their choice of appropriate connectivity metric for different network types. These additional motivations are discussed hereafter.

Efficient computational methods

In Veremyev et al. (2015), an integer programming (IP) model for the distance-based critical node detection problem was proposed. The IP model implementation was only able to compute optimal solutions for small to medium size network instances. Moreover, the model is sensitive to edge-dense topological structures hence struggles computationally as the size of such graphs grows. By exploiting the structure of the problem, we propose new mixed integer programming models for the DCNDP. We demonstrate through computational experiments the competitiveness of the developed models in comparison with the existing compact model. The models proposed in this research can handle all distance-based connectivity objectives defined in Veremyev et al. (2015) and are easily extended to edge weighted graphs as well as the edge deletion version which have not been considered previously in literature.

The distance-based critical node detection problem like many combinatorial optimisation problems is "hard", thus exact solution methods are only able to solve small and medium instances in reasonable time. This opens up the need for the development of heuristic algorithms that would provide sufficiently good solutions in reasonable time. Moreover, solutions from such heuristic algorithms can be used in conjunction with the exact methods either as warm start solution or to routinely guide the optimisation process which could have strong improvement in the solution time. To this end, we propose a new heuristic framework for the distance-based critical node detection problem.

Computational guide for decision makers

An essential aspect in the study of the critical node detection problem is providing appropriate definitions of the concept of critical nodes that lead to feasible solutions and useful outcomes (Borgatti 2006). This entails identifying metrics for determining what nodes (edges) constitute the critical node (edge) set for a given network topology and budget restriction. It is important to note that the variety of studies on the critical node detection problem is due to this issue of metric definition besides the overarching issue of computational burden of solving the problem. While different metrics have been proposed in literature, there is no consensus amongst researchers as to which best defines the critical node set for a given network topology. Existing literature does not provide adequate computational tests on different metrics across a range of network instances such that the suitability of each metric for specific network type might be uncovered. To this end, we provide extensive computational experiments on both real-world and synthetic networks of varied topological structure comparing different metrics for the critical node detection problem.

1.2 Networks and combinatorial optimisation

In this section, we present the crucial elements of the methodological framework underpinning our research. We begin with the nature of optimisation problems in networks with particular emphasis on combinatorial optimisation. We then present some of the basic building blocks of mathematical optimisation as well as solution methods to mathematical optimisation problems.

Researchers in the fields of network analysis and network optimisation study networks and underlying optimisation problems. Network analysis focuses on examination of the internal structures of networks, exploring relationships and associations between objects as well as patterns in networks. One of the most popular and perhaps oldest studies involving network analysis is found amongst the social and behavioural science community. It involves analysis of social networks having its focus on relationships

among social entities such as individuals and groups as well as the patterns and implications of such relationships (Wasserman & Faust 1994, p 3). Social Network Analysis (SNA) is popular for its many interesting applications. For example, it has been employed to explore the spread of diseases as well as to understand diffusion of news, rumours and innovations (Kitsak et al. 2010). Other network analysis studies involve biological networks, for instance protein-protein interactions networks are being studied to understand the nature of different interactions between proteins with a view of improving drug design (Boginski & Commander 2009). Indeed, many problems encountered in various life endeavours can be modeled as optimisation problems in networks. For example finding shortest paths from origin points to destination points, assigning nurses to shifts and assessing vulnerability of networks to disconnection resulting from node or link failures. Network optimisation is concerned with analysing (modelling and solving) optimisation problems in networks. In other words, network optimisation does not only analyse a network against a defined goal or behaviour, but also seeks to do so in an optimal way. Mathematical optimisation entails finding the "best" possible outcome (with regards to some criteria) from a set of available alternatives under given conditions. An optimisation problem involves three basic components: decision variables, set of constraints and an objective function. The decision variables are the elements of a system that we have control over and which we need to decide on, they form the solution to an optimisation problem. This could range from simple decisions such as how many units of a product to transport from facility points to demand points to more complex decisions such as how and when to schedule crew for offshore facility maintenance. The set(s) of constraints constitute conditions that must be satisfied by any solution of the optimisation problem. Depending on the context of the problem, the conditions or limitations range from physical limitations to government policies that businesses must abide by. Examples include production capacity of a plant, warehouse space, health and safety restrictions on maximum hours staff can work per shift and contractual agreements on minimum units to supply to a customer. Also, constraints on decision variables could indicate whether variables must be integer values or continuous. The objective function is the

goal that needs to be actualised. An optimal solution to an optimisation problem is the set of values of decision variables that gives the "best" possible objective function value, where best is either maximum or minimum depending on the problem context.

Many interesting decision problems that occur in networks involve finding a "best solution out of a very large but finite number of possible solutions" (Consoli & Darby-Dowman 2006). Such problems are classified under a branch of optimisation and discrete mathematics called combinatorial optimisation. Examples include Network Flow Problems (e.g. Shortest Path Problem, Minimum Spanning Tree Problem, Maximum Flow Problem), Set Covering Problem, Vertex/ Edge Colouring Problem, Knapsack Problems, Bin-Packing Problem, Network Design Problems (e.g. Survivable Network Design Problem, Steiner Tree Problem), and Traveling Salesman Problem. Because the solution space of combinatorial problems in general is usually very large, most of them are $\mathcal{NP}$-hard, i.e it has not been proven that a polynomial-time algorithm exist to solve them. Hence, for such problems, much research effort is geared towards developing "better" formulations and intelligent algorithms that are specially tailored to solve them. The critical node detection problem on general graph which we study in this research falls under this computationally difficult class of combinatorial problems thus making it an interesting research problem from a computational perspective. It is worth noting that although combinatorial optimisation problems are inherently difficult, there are some easy CO problems whose algorithmic ideas are key in the development of state-of-the-art algorithms for solving the more difficult problems. These "easy" class of network optimisation problems possess some nice structures that make them amenable to be formulated as linear programs and thus solved efficiently. They often appear as subproblems in many complex network optimisation problems. In particular, the shortest path problem and the maximum flow problem constitute subproblems in algorithms for many complex and difficult network problems such as the steiner tree problem, traveling salesman problem and survivable network design problem (e.g, see Hernández-Pérez & Salazar-González (2004), Ljubić et al. (2006), Gouveia et al. (2018)). Our algorithmic development for the distance-based critical node detection problem (see Chapter 3) is derived by repeatedly solving the

shortest path problem as subproblems.

1.2.1 Mixed integer linear programming

In this section, we present the fundamental methods used for modelling and solving combinatorial optimisation problems. These methods form the basis of procedures used to address the problem presented in this thesis. Linear programming is one of the foremost field in mathematical programming with the Simplex algorithm (Dantzig 1951) being fundamental to methods used to solve both linear and mixed integer linear programming problems. A linear programming (LP) model is a mathematical formulation having its objective function and constraint sets as linear functions of the decision variables. An LP is of the form:

LP

$$\min \quad cx \tag{1.1}$$

$$\text{s.t.} \quad Ax \leq b \tag{1.2}$$

$$x \in \mathbb{R}^n_+ \tag{1.3}$$

where c is an n-dimensional row vector, x an n-dimensional column vector of variables, A is an m x n matrix and b an m-dimensional column vector. In general, decision variables in a linear program are assumed continuous however in practice, many application problems require integer variables. For instance, in a nurse rostering problem, you cannot assign 4.2 nurses to a roster. This leads us to mixed integer linear programming model written as:

MILP

$$\min \quad cx + hy \tag{1.4}$$

$$\text{s.t.} \quad Ax + Gy \leq b \tag{1.5}$$

$$x \in \mathbb{R}^n_+, y \in \mathbb{Z}^p_+ \tag{1.6}$$

where G is an m x n matrix, h is a p-dimensional row vector and y is a p- dimensional column vector of integer variables. If all variables are restricted to be integers, then the

formulation becomes an integer programming (IP) model and the special case where only $0-1$ values are allowed then, we have a binary integer programming (BIP) model. Most combinatorial optimisation problems on networks are often formulated as mixed integer or binary integer programs. A given combinatorial optimisation problem can be formulated in different ways giving rise to what is generally known as alternative formulations. By exploring alternative ways of formulating a given problem, sufficient progress can be made towards solving difficult optimisation problem.

Definition 1.2.1. *Given points $x_1, x_2, \ldots, x_k$, a convex combination of $x_1, x_2, \ldots, x_k$ is given by $x = \lambda_1 x_1 + \lambda_2 x_2 + \cdots + \lambda_k x_k$, where $\sum_{i=1}^{k} \lambda_i = 1, \lambda_i \geq 0$.*

Definition 1.2.2. *Given a set $X \subset \mathbb{R}^n$, the convex hull of X denoted by conv(X) is the set of all convex combinations of points in X.*

Definition 1.2.3. *$P \subset \mathbb{R}^n$ is a polyhedron if P is described by a finite set of linear inequalities that is $P = \{x \in \mathbb{R}^n : Ax \leq b\}$*

Proposition 1.2.1. *conv(X) is a polyhedron.*

Proposition 1.2.2. *All extreme points of conv(X) lie in X.*

Propositions 1.2.1 & 1.2.2 indicate that for an integer program represented by IP: $\{\min \quad cx : x \in X\}$, solving the equivalent linear program $\{\min \quad cx : x \in conv(X)\}$ results in integral optimal solution. Thus, by solving the LP over $conv(X)$, the IP is solved. However, the ideal formulation $\{\min \quad cx : x \in conv(X)\}$ for an IP is usually practically prohibitive since the description of the $conv(X)$ requires an exponential number of inequalities which are not easy to characterise. Since, the difficulty in solving an integer programming problem is directly linked to the number of constraints in a formulation, formulations with more constraints are likely to be more difficult to solve (leading to more computational time) than those with fewer constraints. Thus, for many large combinatorial optimisation problems, one pragmatic approach is to construct formulations whose constraint set consists of many trivially satisfied constraints with only a few active constraints. These constraints which are known as lazy constraints are

dynamically generated only when violated hence the formulation size eventually becomes much less. Another advantage of the lazy constraint formulation is that since the initial model size is small, the equivalent linear programming relaxation is easily solved thus providing quick dual bounds. The concepts of bounds are useful in proving optimality and are crucial components of the exact algorithms discussed hereafter. We refer the reader to Wolsey (1998) and Wolsey & Nemhauser (1999) for a review of formulations and methods for solving integer programming problems.

Definition 1.2.4. *Consider the integer program:*

$$min\{cx : Ax \leq b, x \in \mathbb{Z}_+^n\}$$

Then, the linear programming relaxation to the IP is given as

$$min\{cx : Ax \leq b, x \in \mathbb{R}_+^n\}$$

Branch-and-Bound algorithm

One of the popular global optimisation algorithms for integer and mixed integer programming problems is the branch-and-bound algorithm (Land & Doig 1960). The algorithm is based on the divide and conquer principle wherein a given problem is partitioned into smaller subproblems which when solved and aggregated together provide solution to the original problem. The basic idea of the branch-and-bound algorithm is as follows:

(i) partition the feasible set into smaller convex sets,

(ii) find lower and upper bounds for each subset

(iii) form global lower and upper bounds;

(iv) if the global lower and upper bounds are close enough quit, else, refine the partition and repeat

The branch-and-bound algorithm relies on the computation of lower and upper bounds since the bounds provide information to prune the branches thus reduce the computational cost of enumerating the tree.

Proposition 1.2.3. *Let $X = X_1 \cup \cdots \cup X_K$ be a decomposition of X into smaller sets, and let $z^k = min\{cx : x \in X_k\}$ for $k = 1, \ldots, K$, $\overline{z}^k$ be an upper bound on z^k and $\underline{z}^k$ be a lower bound on z^k. Then, $\overline{z} = min_k \overline{z}^k$ is an upper bound on z and $\underline{z} = min_k \underline{z}^k$ is a lower bound on z .*

Since LPs are more like IPs without the integrality restrictions, linear programming relaxations provide dual bounds when solving IPs with the branch-and-bound algorithm. The branch-and-bound algorithm solves the LP relaxation of an IP. More specifically, at the root node, the LP relaxation $X_0^{LP} : \{\min \quad cx : Ax \leq b, x \in \mathbb{R}^n\}$ of a given IP is solved. This gives rise to a lower bound $\underline{z}$. As no feasible solution to the IP exist yet, we take as upper bound $\overline{z} = \infty$. Ideally, $-\infty \neq \underline{z}$ and $\underline{z} < \overline{z}$, so the algorithm needs to divide the feasible region (branch) by selecting a branching variable usually an integer variable with the most fractional value say x_i. This splits the problem into two subproblems about the fractional value x_i^* of the branching variable yielding two new LP problems:

- $X_1^{LP} : \{\min \quad cx : Ax \leq b, x_i \geq \lceil x_i^* \rceil, x \in \mathbb{R}^n\}$
- $X_2^{LP} : \{\min \quad cx : Ax \leq b, x_i \leq \lfloor x_i^* \rfloor, x \in \mathbb{R}^n\}$

Each of the subproblems is evaluated for their lower bound $\{\underline{z_1}, \underline{z_2}\}$ and upper bounds $\{\overline{z_1}, \overline{z_2}\}$. Based on the bounds of the subproblems, the global lower and upper bounds are updated respectively as follows: $\underline{z} = \min\{\underline{z_1}, \underline{z_2}\}$ and $\overline{z} = \min\{\overline{z_1}, \overline{z_2}\}$. Using these bounds, we can then prune the branch-and-bound tree thereby reduce the number of further decomposition and hence subproblems to be examined. If the optimal solution to the LP problems X_i^{LP} is integer, then the associated integer problem is solved hence X_i^{LP} is pruned by optimality. If $\underline{z}_i \geq \overline{z}$, then the branch X_i of the tree is pruned by bound. The reasoning follows from the fact that $\overline{z}$ is an upper bound to the optimal objective value z of the original integer programming problem as seen in proposition 1.2.3. Thus, no optimal solution can lie in the set X_i. If there exist X_i for which none of the conditions for pruning applies, the branch is considered active and branching is done. The algorithm continues the search in this manner until no active problem exists and the global upper

bound is returned as the optimal value with the corresponding solution as an optimal solution. A feasible solution can also be obtained using a heuristic algorithm to kick start the branch-and-bound process as well as to provide primal (upper) bounds periodically all through the branch-and-bound process.

Branch-and-Cut algorithm

From the description of the branch-and-bound algorithm, we can see that obtaining good primal and dual bounds (values that are close to the optimal objective value) would help the algorithm to converge faster. As have been noted, LP-relaxation provides dual bounds to an integer program. However, bounds obtained from the LP-relaxation solution might be far from the optimal objective value which makes the branch-and-bound tree slower. The cutting plane method provides a way to strengthen the bounds of LP through an iterative addition of valid inequalities that are satisfied by all integer feasible solutions but are violated by the current LP-relaxation solution. The idea is that the addition of generated inequalities (or cuts) cuts off many fractional solutions. This is with the hope that the feasible region of the LP relaxation is reduced to one whose extreme points are integer and hence solving the resultant LP solves the IP (Huang et al. 2009). However, it could require an exponential number of steps to encounter an integral extreme point, thus the cutting plane method on its own is not a very efficient way to solve difficult integer programs in practice. In practice, the branch-and-bound algorithm is usually combined with the cutting plane method to solve difficult integer programming problems. This is the so-called branch-and-cut algorithm which is a branch-and-bound method in which cutting planes are generated all through the branch-and-bound tree (Wolsey 1998). The underlying idea of the branch-and-cut algorithm is to work towards getting tighter dual bounds in each node of the tree by adding appropriate cutting planes thereby reduce the number of subproblems (nodes) in the search tree. In solving many difficult problems, the implementation of a branch-and-cut algorithm also involves the construction of good feasible solutions based on primal heuristics that incorporate information from LP-relaxation solutions.

Thus, successfully solving combinatorial optimisation problems in practice requires an intelligent combination of formulation of good integer programs, fast feasible solutions, cutting planes etc.

Research Question

Based on the research issues and motivations introduced with respect to the critical node (edge) detection problem, our research question is defined thus: *How can we efficiently identify nodes (edges) of networks of varied sizes and topological structures that are critical to network connectivity objectives relevant for real-world networks?*

Research contributions

The contributions of our research consist of the following:

- New mathematical programming models and exact algorithm for various distance-based connectivity objectives which are relevant to real-life application. Our models are more computationally efficient than the existing compact model particularly as the input network grows in size and cohesiveness.
- Efficient heuristic that provides good solutions in reasonable time particularly for the challenging problem instances unsolved by exact methods.
- Extensive empirical results that compare different connectivity metrics which would provide useful insights to decision makers in choosing appropriate metric for their application contexts.

1.3 Outline of thesis

The rest of the report is organized as follows. In Chapter 2, a comprehensive review of literature on the topic of node and edge deletion problems in networks is presented. In Chapter 3, a formal description of the distance-based critical node detection problem

is presented along with the integer programming model by Veremyev et al. (2015). We then present our proposed mixed integer programming models and exact branch-and-cut algorithm for the edge-unweighted distance-based critical node detection problem. Chapter 4 focuses on heuristic algorithm for the distance-based critical node detection problem. In Chapter 5, we present generalisations of the distance-based critical node detection problem to edge-weighted distances where we propose two mixed integer programming models. We also present models for the edge-deletion version (which we term the distance-based critical edge detection problem). Conclusion and future research directions are presented in Chapter 6.

Chapter 2

Literature Review

In this chapter, we present a critical review of existing research. We begin with studies on node and edge deletion problems in networks, where we look at different contexts and applications for both node and edge deletions. We then focus on studies on node deletions with specific emphasis on critical node detection problem which is the problem that seeks nodes in network whose removal minimise some connectivity measure in the residual network. We review studies on the complexity of the critical node detection problem as well as different classification and variants that have appeared in literature. We conclude the review with overview of solution methods developed for the critical node detection problem.

2.1 Node and edge deletion problems

Studies on node and edge deletion problems can be traced back to the late 1970's with the works of Yannakakis (1978) and Krishnamoorthy & Deo (1979). Given a graph property π, the general node (edge) deletion problem is to identify the minimum number of nodes (edges), whose deletion results in a subgraph satisfying property π (Yannakakis 1978). Krishnamoorthy & Deo (1979) proved the NP-completeness of node-deletion problems for 17 different graph properties including the induced subgraph being complete, a tree, planar, and bipartite amongst others. This complexity result was generalised by Lewis &

Yannakakis (1980) for a broad class of properties namely properties that are hereditary on induced subgraphs including requiring that the induced subgraph be connected. For the edge version, the authors showed that same generalisations do not hold, however for many common properties (e.g. planar, line-graph) the edge-deletion problem is $\mathcal{NP}$-complete. Subsequent studies on node and edge deletion problems built upon these complexity results with particular focus on the property of "connectedness" in the induced subgraph. The emphasis on the property of "connectedness" is largely motivated by the importance of connectivity in many real-world problems that are modelled on networks. Indeed many network problems are concerned with the issue of whether or not nodes (or specific node pairs) are connected as well as the extent (measured in terms of multiple paths availability, shortest path length) of their connection.

Network interdiction is one such network problem with interesting application in security and defense operations especially for combating drug trafficking. In its simplest form, network interdiction problem can be seen as an adversary-interdictor game which takes the form of the well-known max flow-min cut theorem (Ford Jr & Fulkerson 1962). An adversary seeks to maximise the flow of a commodity from source node s to destination node t in a directed network while the interdictor tries to break (delete) arcs in the network in order to eliminate all possible paths for the enemy. Associated with each arc (i, j) is a resource requirement r_{ij} required to break the arc and the interdictor wishes to use minimum total resource to disrupt all $s - t$ paths (Wood 1993). Here, we see the interdictor's aspect of the problem as a classical edge deletion problem involving the minimisation of the enemy's maximum flow achievable along unbroken edges of the network using no more than R available resources. Other kinds of network interdiction problems have also been studied. The shortest path network interdiction where the goal of the interdictor is to maximise the shortest path length of an attacker's origin-destination pair (Israeli & Wood 2002, Bayrak & Bailey 2008). Other versions include partial arc interdiction (Israeli & Wood 2002, Sadeghi et al. 2017), stochastic interdiction (Cormican et al. 1998, Morton 2010)) and multi-objective network interdiction (Rocco & Ramirez-Marquez 2010, Rocco et al. 2010). For a review on models and

algorithms for network interdiction problems, we refer the interested reader to the recent survey by Smith & Song (2020).

Node and edge deletion problems have appeared in the context of network vulnerability and risk assessment following investigations of responses of complex networks to failures and attacks (Albert et al. 2000, Holme et al. 2002). These studies revealed that real-life networks while being robust to random failures are vulnerable to targeted attacks. While these studies provided useful insights as to the behavioural changes and hence vulnerability of complex networks to different events (failures and attacks), they did not seek to identify those nodes and/or edges that made the networks most vulnerable to attacks. This is important since the goal of vulnerability assessment does not just lie in knowing whether or not a network is vulnerable but in identifying the main sources of its vulnerability so as to fortify them against any disruption. This brings us to a very important class of node and edge deletion problems namely the critical node detection problem (CNDP) introduced by Arulselvan et al. (2009) which we now focus in the rest of this review chapter.

2.2 Critical node detection problem

The critical node detection problem comes as a natural problem in vulnerability and risk assessment of complex networks. Network vulnerability relates to the degradation of network performance due to failures of nodes (edges) in the network. Critical nodes are those whose deletion significantly degrades the overall network connectivity. Thus the performance metric used to assess network vulnerability in the study of the critical node detection problem is that of connectivity. Defining appropriate connectivity metric is of great importance in determining what nodes are critical in a given network. Indeed it has been noted that an essential aspect of the study of the critical node detection problem is the definition of appropriate metric (Borgatti 2006, Chen 2015). This is because the defined objective metric affects the feasibility of the associated problem as well as the usefulness of obtained solutions (Borgatti 2006). Thus, defining an appropriate metric

helps to describe how the resultant subgraph must be disconnected once the nodes have been deleted leading to an accurate determination of the concept of criticality for a given input problem (Lalou et al. 2018). With respect to network connectivity, different objective metrics have been defined for the study of the critical node detection problem. These can be grouped into two broad classes: fragmentation-based critical node detection problem and distance-based critical node detection problem.

2.2.1 Fragmentation-based critical node detection problem

Network fragmentation is the underlying goal of earlier set of studies on critical node detection problem. The goal of network fragmentation is to break the input network into as many components as possible since nodes in different components cannot establish a connection. Fragmentation-based critical node detection problem finds application in a variety of problems such as in the dismantling of terrorist network by breaking down communication between individuals in the network (Arulselvan et al. 2009). Variants of the fragmentation-based critical node detection problem that have appeared in literature are hereby reviewed.

Critical node problem (CNP)

The pioneer study on the critical node detection problem is the critical node problem (CNP) proposed by Arulselvan et al. (2009). With the goal of network fragmentation in mind, the authors defined the problem as follows:

> "Given a graph $G = (V, E)$ and an integer B, the objective of the critical node problem (CNP) is to find a set of B nodes whose deletion results in the maximum network fragmentation in other words minimise the pair-wise connectivity between the nodes in the B-vertex deleted subgraph." (Arulselvan et al. 2009)

The authors established the $\mathcal{NP}-$completeness of the decision version of the problem. They proposed a linear integer programming formulation which consists of $O(n^3)$ triangular-inequality

constraints. Due to the computational limitation of the proposed model, they proposed a heuristic that exploits the structure of the maximum independent set problem to solve instances of up to 150 nodes. The computational limitation of the IP model proposed by Arulselvan et al. (2009) gave rise to further research efforts that were geared towards developing more efficient models and algorithms as well as identifying polynomial solvable classes of the CNP. Among them are the works of Di Summa et al. (2011), Di Summa et al. (2012), Addis et al. (2013) and Veremyev, Boginski & Pasiliao (2014). In particular, Di Summa et al. (2011) studied the weighted version of the CNP over trees and proved that the CNP over trees in general is still $\mathcal{NP}$-complete. However, they show that for unit connection costs that is $w_{ij} = 1 \ \forall (i,j) \in E$, the CNP on trees is polynomial solvable and proposed a dynamic programming algorithm for it. They also proposed an enumeration scheme for trees with unit node weights and non-negative connection costs. However, computational test was limited to instances of up to 100 nodes due to space complexity of the algorithm.

In Addis et al. (2013), the authors studied the unweighted version of the CNP on some graph classes and established the hardness of the CNP over split and bipartite graphs. Using a tree decomposition of the input graph, they proposed a dynamic programming algorithm through which they showed that the CNP on graphs with bounded treewidth is solvable in polynomial time.

In Di Summa et al. (2012), a path-based integer linear programming (ILP) model for the CNP was proposed. Although the number of constraints in the path-based ILP was potentially exponential, the authors showed that they could be separated in polynomial time by solving an all-pairs shortest path problem on an auxiliary weighted graph. The authors derived valid inequalities on special structures such as cliques and cycles which were implemented in a branch-and-cut framework to solve instances of the Barabasi-Albert and uniform random graphs of up to 100 nodes. They also proposed a quadratic reformulation for the problem although no empirical result was presented.

The current state-of-art IP model for the critical node problem on sparse graphs is attributed to Veremyev, Boginski & Pasiliao (2014). The authors proposed a compact

IP model that requires fewer constraints in comparison to the model by Arulselvan et al. (2009) along with some reformulations and valid inequalities to enhance the proposed model. They were able to solve real-world sparse networks of sizes 10 times larger than have been solved by previous exact methods. Their computational experiments also show CPU time for their formulation to be 1000 times faster than prior existing models.

Some researchers have also studied the edge version of the critical node problem where edges (arcs) rather than nodes are deleted. In Matisziw & Murray (2009), an alternative flow interdiction model was developed to assess the availability of connection between source and destination (s, t) pairs upon failure of certain number of arcs. The model incorporates path aggregation constraints to overcome the computational burden of path enumeration inherent in previous related models. Computational experiment based on the formulation was carried out on the Ohio interstate highway network having 23 nodes. Shen et al. (2013) showed that the edge version of the CNP which they termed critical link disruptor problem is $\mathcal{NP}$-complete on general graphs as well as on special graphs specifically unit disk and power-law graphs. They proposed an LP-based rounding algorithm which was implemented on both real-world and synthetic networks. The stochastic and robust versions of the critical node problem have also been studied for example in Naoum-Sawaya & Buchheim (2016) and Hosteins & Scatamacchia (2020).

CC-CNP and $\beta-$disruptor problem

The classical critical node detection problem which minimises total pairwise connectivity is suitable when the decision maker has prior information on the size of the critical node set or is constrained by a given budget. However, when assessing network vulnerability, it might be desirous to ascertain the number of node (or edge) failures necessary to cause a certain level of disruption in a network. It might also be the case that specifying a budget value might be too restrictive for some network topological structures thereby hampering the possibility and ease of obtaining a feasible solution. Thus, we can direct the objective focus on identifying the minimum set of critical nodes whose deletion achieves certain levels of fragmentation (pairwise disconnections) in the residual graph. By altering the

fragmentation levels, the decision maker can assess the robustness of a network. This is the motivation behind two additional variants of the fragmentation-based critical node detection problem namely cardinality constrained critical node problem (CC-CNP) and β- disruptor problem. In the former, the objective is to minimise the number of nodes whose deletion results in connected components of size less or equal to some specified value. The latter deals with both node and edge deletions and seeks to minimise the number of deleted nodes (or edges) such that the proportion of connected node pairs in the remaining subgraph is bounded by a threshold β. Like the CNP, the cardinality-constrained CNP is $\mathcal{NP}$-complete on general graphs as shown in Arulselvan et al. (2011). Subsequently, the authors proposed a genetic algorithm with local search procedures for the CC-CNP. A more compact model that provides exact optimal solutions for larger real-world sparse network instances was proposed by Veremyev, Boginski & Pasiliao (2014). Similar to the CC-CNP, the β-disruptor problem is $\mathcal{NP}$-complete for both node and edge versions (Dinh et al. 2012). A closely related version to the CC-CNP is the component-cardinality-constrained critical node problem (3C-CNP) defined and studied by Lalou et al. (2016) as well as Lalou & Kheddouci (2019).

MaxNum and MinMaxC

Alternative fragmentation objectives that have also appeared in literature are those that respectively maximise the number of connected components (MaxNum) and minimise the largest component size (MinMaxC). The aim of these objectives is to avoid network fragmentation that results in many isolated nodes with a giant component in the subgraph. This is because such resultant subgraph cannot be seen as effectively disconnected in practical contexts such as covert network since members of the giant component can still communicate and co-ordinate a terrorist attack. Shen et al. (2012) established the $\mathcal{NP}$-hardness of both the MaxNum and MinMaxC on general graphs. The authors developed MIP formulations and derived valid inequalities as well as objective bounds for the MIPs based on dynamic programming solutions of k-hole subgraphs of the input graph. They also introduced a new CNP variant which they termed the MaxMinLR.

The goal of this less-studied variant is to maximise the minimum cost required to reconnect the graph after node deletions given a set of edge construction cost. In Shen & Smith (2012), the authors studied the MaxNum and MinMaxC on special graphs namely trees and series-parallel graphs. They developed a polynomial-time dynamic programming algorithm to solve these problems on randomly generated tree instances having a maximum of 12 levels and average of 4102 nodes. We refer the interested reader to the survey by Lalou et al. (2018) for a more detailed review on fragmentation-based critical node detection problems.

It is worth noting that the aforementioned studies on the fragmentation-based critical node detection problem are restricted to individual node removals and hence do not consider the structural relationships between members of the critical node set. A new paradigm that generalises the fragmentation-based critical node detection problem is the so-called critical node structure detection problem where the set of nodes to be deleted form a specific structure (Walteros et al. 2019). Thus, the underlying problem becomes further constrained depending on the desired structure of the critical node set such as cliques and stars giving rise to the new terms critical cliques and critical stars.

2.2.2 Distance-based critical node detection problem

Fragmentation-based critical node detection problem have been well studied in literature as seen by the volume of research output on this problem, all seeking to develop more efficient model and solution methods for larger instances. However, as pointed out earlier in Section 1.1, the CNDP whose goal is network fragmentation may not be appropriate for some network applications where the extent of connection is important. This is because fragmentation-based CNDP merely cares about the existence of a path between pairs of nodes. Hence, in the objective function, equal level of connectivity is assigned to any pair of nodes with a connecting path regardless of the length of the shortest path connecting them. Moreover, depending on the topology of the input graph, for instance, in dense networks where fragmenting the network under imposed budgetary constraints is infeasible, achieving a sufficiently large increase in pairwise distances can be seen as

practical disconnection. Thus, to address such application contexts, Veremyev et al. (2015) introduced a new variant of the critical node detection problem that accounts for actual pairwise distances. This is the distance-based critical node detection problem (DCNDP), which minimises distance connectivity objectives by incorporating into its objective a distance function $f(d)$ for the shortest path length between any pair of nodes. The authors specified 5 distance-based connectivity objectives otherwise referred to as classes of the distance-based critical node detection problem namely:

(i) Minimise the number of node pairs connected by a path of length at most k

(ii) Minimise the Harary index (equivalently, the efficiency)

(iii) Minimise the sum of power functions of distances

(iv) Maximise the generalized Wiener index (equivalently, the characteristic path length)

(v) Maximise the shortest path length between nodes s and t

They also proposed a generic integer linear programming model that admits all 5 distance connectivity objective as well as an exact algorithm that iteratively resolves simpler versions of the IP model. The first class of the DCNDP which minimises the number of node pairs connected by path of length k reduces to the classical critical node problem when $k = n - 1$. Since, the classical CNP is $\mathcal{NP}$-complete on general graphs, it therefore follows that class (i) of the DCNDP on general graphs is equally $\mathcal{NP}$-complete. Building on existing complexity studies of the CNP, Aringhieri et al. (2019) analysed the complexity of classes (i), (iii) and (iv) of the DCNDP on special graph classes such as paths, trees and series-parallel graphs. They observed that the presence of a distance-based cost function in the objective makes the complexity analyses of DCNDP more complicated when compared to the simple pairwise-connectivity CNP. Thus, under certain conditions such as unit node deletion cost and/or non-negative edge weights, they identified polynomial and psuedo-polynomial solvable classes of DCNDP. They proposed dynamic programming algorithms for such classes however no empirical experiment was presented. In Hooshmand et al. (2020), a complementary MIP formulation and a Benders

decomposition algorithm were proposed for the DCNDP class (iv). More recently, in an independent study by Salemi & Buchanan (2020), new integer programming formulations for the first distance based connectivity objective (i) were proposed. Their solution technique uses warm-start solutions obtained from a heuristic as well as a preprocessing procedure that identifies "non-critical" nodes for which corresponding variables are fixed to zero.

Variants of both the fragmentation-based and the distance-based critical node detection problems along with relevant literature references are summarised in Table 2.1. Unlike fragmentation-based critical node detection problem, studies on the distance-based critical node detection problem have been quite limited partly due to its recent definition. Our research was aimed towards exploring this new frontier of node and edge deletion problems primarily from a computational perspective so as to provide more efficient methods to solve the problem.

2.2.3 Multi-objective critical node detection problem

It is worthy to note that there is no consensus amongst existing studies as to the most suitable metric for studying the critical node detection problem since application contexts influence the suitability of each metric. Thus, some researchers have attempted to capture more than one metric in their objective giving rise to what can be classified as multi-objective critical node detection problem. The first study in this direction is that of Veremyev, Prokopyev & Pasiliao (2014) who were motivated by the failure of fragmentation-based CNDP metric to capture cohesiveness property in the residual graph. Thus they proposed a weighted bi-objective integer programming model which minimises both connectivity and cohesiveness metrics. The connectivity and cohesiveness metrics were assumed to be increasing functions of sizes of connected components and node degrees respectively. Thus, the model could be seen as a unifying model for fragmentation-based CNDPs with the added advantage of incorporating a cohesiveness term in its objective function. A striking conclusion drawn from this study is that standard MIP solvers are able to handle reasonable size of real-world instances depending

CNDP Classification	CNDP Variant	Objective	Citation
Fragmentation-based CNDP	CNP	Minimise pairwise connectivity by deleting a subset of nodes of specified cost B	Addis et al. (2016), Aringhieri et al. (2016a,b), Arulselvan et al. (2009), Boginski & Commander (2009), Di Summa et al. (2011, 2012), Matisziw & Murray (2009), Myung & Kim (2004), Shen et al. (2013), Purevsuren et al. (2016), Li et al. (2019), Ventresca & Aleman (2014, 2015), Ventresca (2012), Veremyev, Boginski & Pasiliao (2014), Veremyev, Prokopyev & Pasiliao (2014), Zhou et al. (2018), Zhang et al. (2020), Pullan (2015)
	CC-CNP	Limit the maximal component size to a given bound by minimising the total cost of node deletion	Aringhieri et al. (2016a), Arulselvan et al. (2011, 2007), Lalou et al. (2016), Veremyev, Boginski & Pasiliao (2014), Zhou et al. (2018)
	MinMaxC	Minimise the largest component size by deleting a subset of nodes of specified cost B	Shen & Smith (2012), Shen et al. (2012), Veremyev, Prokopyev & Pasiliao (2014), Aringhieri et al. (2016a), Nguyen et al. (2013)
	MaxNum	Maximise the number of connected components by deleting a subset of nodes of specified cost B	Shen & Smith (2012), Shen et al. (2012), Veremyev, Prokopyev & Pasiliao (2014), Aringhieri et al. (2016a), Ventresca et al. (2018)
	β-disruptor problem	Bound pairwise connectivity to some specified threshold beta by minimising the total cost of node(edge) deletion	Aringhieri et al. (2016a), Dinh et al. (2012), Dinh & Thai (2015)
Distance-based CNDP	DCNDP class 1	Minimise the number of node pairs connected by a path of length at most k by deleting a subset of nodes of specified cost B	Veremyev et al. (2015), Aringhieri et al. (2019), Alozie et al. (2021)
	DCNDP class 2	Minimise the Harary index or efficiency by deleting a subset of nodes of specified cost B	Veremyev et al. (2015), Aringhieri et al. (2019), Alozie et al. (2021)
	DCNDP class 3	Minimise sum of power functions of distances by deleting a subset of nodes of specified cost B	Veremyev et al. (2015)
	DCNDP class 4	Maximise the Wiener index by deleting a subset of nodes of specified cost B	Veremyev et al. (2015), Aringhieri et al. (2019), Hooshmand et al. (2020)

Table 2.1: Variants of the CNDP

on the following three factors viz: (i) the type of objective function (ii) the specified budgetary limitations (iii) the topology of the input graph. This buttresses the motivating points for our research particularly on the need to develop more efficient exact and heuristic solution methods for the CNDP especially the distance-based objectives where computational studies are limited. In Ventresca et al. (2018), a bi-objective CNDP that seeks to maximise the number of connected components in the residual graph while minimising the variance of the component sizes was proposed. Through computational comparisons of 6 common multi-objective evolutionary algorithms on 16 synthetic benchmark problem instances, the authors identified the non-dominated sorting genetic algorithm II (NSGAII) as the most promising algorithm. The proposed bi-objective CNDP might be seen as extending fragmentation-based CNDP since it is only concerned with sizes of connected component with no recourse to cohesiveness unlike the one proposed by Veremyev, Prokopyev & Pasiliao (2014). Other bi-objective approaches are those by Li et al. (2019) and Zhang et al. (2020) both of which seek to simultaneously minimise pairwise connectivity of the induced graph and the cost of removing the critical nodes. Thus both studies are primarily focused on the classical fragmentation-based CNDP with the added opportunity to optimise the cost of node deletion which is important when the budget on the critical node set is not known apriori.

2.3 Solving the critical node detection problem

We now review solution methods for the critical node detection problem existing in literature. Specifying a mathematical programming formulation for a combinatorial optimisation problem is the first step towards understanding the structure of the problem and analysing its complexity. It is upon such analyses and understanding that algorithm development is built. Solution methods for the critical node detection problem on complex networks can be classified into either exact or heuristic algorithms.

2.3.1 Exact algorithms for the critical node detection problem

Exact approaches for the critical node detection problem comprise mainly solving proposed integer and mixed integer programming models using default algorithms in popular optimisation solvers such as CPLEX (Cplex 2009) and Gurobi (Gurobi Optimization 2018). Examples of such studies include those by Arulselvan et al. (2009), Veremyev, Prokopyev & Pasiliao (2014) and Veremyev, Boginski & Pasiliao (2014). Default algorithms in commercial solvers use generic cutting planes which might not be effective since they lack understanding of the problem structure. By exploiting special structures of the different variants of the critical node detection problem, researchers have devised efficient cutting planes which are employed within a branch-and-bound framework. In Di Summa et al. (2012), valid inequalities based on cliques and cycles were proposed for the classical critical node problem. In Veremyev et al. (2015), an exact truncate-and-resolve algorithm which iteratively solves a series of simpler integer programs was proposed for the distance-based critical node detection problem. Benders decomposition approach was used for both the classical critical node detection problem (Naoum-Sawaya & Buchheim 2016) and for a class of the distance-based critical node detection problem (Hooshmand et al. 2020). Solution approaches for the critical node detection problem on special graph structures such as trees are largely based on dynamic programming. Examples of these include Di Summa et al. (2011), Shen et al. (2012), Shen & Smith (2012) and Aringhieri et al. (2019).

2.3.2 Heuristic algorithms for the critical node detection problem

Due to the computational complexity of the critical node detection problem, research have been advanced in the direction of heuristics to provide good solution to larger instances in reasonable time. In Ventresca (2012), a population based incremental learning (PBIL) heuristic and simulated annealing heuristic were proposed to solve instances of the classical critical node problem with size ranging between $250 - 5000$ nodes. Ventresca & Aleman (2015) also proposed a greedy algorithm that is based on a modified depth first search for the classical critical node problem (minimisation

of pairwise connectivity). Their algorithm which runs in $O(k|V| + |E|)$ time was able to compute set of critical nodes for graphs with up to $23,133$ nodes. Their algorithm provided solutions with better (smaller) objective function values in comparison with popular centrality-based algorithms such as Degree and PageRank although at the cost of longer running time. Aringhieri et al. (2016*b*) developed solutions for the classical CNP based on an improved variable neighbourhood search (VNS). The new neighbourhood structures were shown to be more computationally efficient than traditional two node exchange neighbourhood structure. A hybrid constructive heuristic was developed in Addis et al. (2016) for the classical CNP. The heuristic combines greedy algorithms and local search techniques by first identifying a vertex cover then modifies the identified cover for objective function improvements until the budgetary constraint is satisfied. A genetic algorithm for the classical CNP and its cardinality-constrained variant was also proposed by Aringhieri et al. (2016*a*). A Greedy Randomised Adaptive Search Procedure (GRASP) with Path Relinking (PR) mechanism was proposed for the classical CNP by Purevsuren et al. (2016). In comparison with 3 existing CNP heuristics, the GRASP-PR algorithm outperformed two of the algorithms (PBIL and simulated annealing) developed by Ventresca (2012) for all 16 benchmark instances. It also provided better solutions than its competitor VNS (Aringhieri et al. 2016*b*) for 13 out of the 16 test instances. In Zhou et al. (2018), the authors developed a memetic algorithm for the classical CNP and demonstrated extension of the framework to the cardinality-constrained CNP. The algorithm maintains a population of promising solutions generated through a combination of double backbone-based crossover, a component-based neighbourhood search procedure and a rank-based pool updating strategy. The double backbone-based crossover procedure generates promising offspring solutions by inheriting both common and exclusive elements of its parent solutions. The component-based neighbourhood uses a node weighting technique for node selection from a reduced neighbourhood (large connected components) thus making the neighbourhood search more efficient. Computational experiments on 42 benchmark instances (26 real-world and 16 synthetic instances) show the effectiveness of the proposed memetic algorithm over existing ones including an iterative local search

algorithm (Addis et al. 2016), variable neighbourhood search heuristic (Aringhieri et al. 2016*b*) and genetic algorithm (Aringhieri et al. 2016*a*). In particular, the proposed memetic algorithm matches 18 previous best-known upper bounds and discovers 21 new ones, falling short in only 3 of the instances. There is yet no existing heuristic algorithm for any of the classes of the distance-based critical node detection problem hence our research (Chapter 4) is aimed at addressing this gap.

2.4 Critical node detection problem: application and related problems

We end this chapter with a review of interesting applications of the critical node detection problem as well as network problems that are closely related to the critical node detection problem.

2.4.1 Applications of critical node detection problem

As have been previously noted, research on the critical node detection problem is inspired by a variety of areas in which it can be applied. Some of these areas are reviewed.

Computational Biology

The critical node detection problem has been applied to study the interaction of proteins in biological networks. Biological organisms such as bacteria consist of sets of interconnected proteins whose interactions form protein-protein interaction networks with nodes as proteins and interactions as edges (Lalou et al. 2018). Thus, critical nodes in the context of protein-protein interaction networks are the proteins that are important to the connectivity of the network. In Boginski & Commander (2009), the authors employed the cardinality-constrained critical node problem (CC-CNP) variant to identify critical proteins whose removal would destroy the primary interactions leading to a neutralisation of harmful organisms such as bacteria. The authors proposed that the application of the CNDP in protein-protein interaction networks is potentially useful in drug design. This

was later investigated by Tomaino et al. (2012) in the context of human protein-protein interaction network. Applying the classical CNP variant on a human cancer protein network, critical proteins were identified as hub proteins that are responsible for the overall connectivity of the graph whose mutation can lead to cancer and other diseases.

Contagion control

Another important area where the critical node detection problem finds application is in contagion control. Connectivity is a very crucial phenomenon in the outbreak of infectious diseases, computer virus or rumours within human or computer networks. Thus, breaking or limiting connections between nodes of the networks is necessary to contain the spread of these undesirable events. The classical CNP has been proposed as an efficient strategy for immunisation of populations against diseases (Arulselvan et al. 2009). Since vaccinating an entire population is impracticable, immunising or isolating the identified critical nodes provides a cost-effective way to halt infection propagation. The authors maintained that the proposed strategy is efficient and better than previously existing immunisation strategies. Because existing CNP models consider all nodes to be from the same set, they are not well suited for direct application to the context of infection control. Hence, in Nandi & Medal (2016), the authors gave specific attention to the problem of minimising the spread of infection through link removal and divided the node set into susceptible and infected nodes. They proposed four different connectivity-based network interdiction models. The four models which were formulated as mixed integer linear programs minimise

- the number of connections between infected and susceptible nodes
- the number of susceptible nodes having one or more connections with infected nodes
- the total number of paths between infected and susceptible nodes
- the total weight of the paths between infected and susceptible nodes

The motivation for link removal is based on the fact that in some network application, node removal is more expensive. For example, it is more feasible and less expensive to temporarily cancel flights between two specific airports than to completely shut down an entire airport during a global pandemic. A combination of both node and link removal have also been proposed for infection control (He et al. 2011). Also, for some newly developing infection for which vaccines are unavailable and complete isolation is not feasible or desirable, the distance-based CNDP variants could be potentially useful as a means of limiting distance between members of a population. The 2 metres social distancing policy introduced in the United Kingdom during the 2020 coronavirus pandemic could be viewed as an intuitive application of distance-connectivity measure in slowing the spread while allowing for "essential" activities. This is also relevant for contagions that require "very close" contact with infected persons to propagate across the network. The cardinality constrained version (CC-CNP) was proposed to be a potential model for the case of complex contagion spreading which requires interaction with atleast $t > 1$ infected individuals for an individual in the network to be infected (Lalou et al. 2018). In which case, given a cardinality constraint t on size of components, deletion of the critical nodes leaves each component with at most t nodes either all of which are already infected or at least one of the t nodes is not infected and thus the contagion cannot spread.

Disaster vulnerability assessment and management

Economic and social activities are heavily dependent on operations of network infrastructures such as highways, electrical power grids, telecommunication systems. The loss or failure of one or a few of the facilities can result in wide-range service disruption. One way of curtailing such disruptions is to identify the critical elements of these networked systems in order to fortify them against attack. This helps disaster management planners to reduce the worst-case risk of a disaster. Moreover, identifying the critical facilities can reduce the burden of planning for disaster recovery since budgetary resources are limited. The classical CNP has been applied in the context of disaster vulnerability assessment

and management to identify network facilities that are most vital to system flow. For instance, models for arc deletion versions of the CNP have been proposed as a means of identifying the facilities (roads) of a networked system associated with the worst-case impact to system flow (Myung & Kim 2004, Murray et al. 2007, Matisziw & Murray 2009). The enhanced path model by Matisziw & Murray (2009) was applied to the Ohio's interstate system to assess the vulnerability of its trucking activity (as a measure of system performance) to interstate disruption. Most of these studies that apply the CNDP to vulnerability assessment of critical infrastructures assume that system flow is ensured by availability of an $s-t$ path, hence use CNDP models that are based on fragmentation. The Distance-based critical node detection problem can be used to address network vulnerability where the length of available paths is equally important. A classical example is in telecommunication network where quality of service (usually modelled by hop distance) is a crucial element of service delivery which internet service providers consider in vulnerability assessment (Elsayed 2004, Gouveia & Leitner 2017).

2.4.2 Network problems related to critical node detection problem

Key player problem

The key player problem (Borgatti 2006) in social network is the problem of identifying actors/individuals that are important for the network. According to Borgatti (2006), there are fundamentally two classes of key player problem namely key player problem/negative (KPP-Neg) and key player problem/positive (KPP-Pos). KPP-Neg is defined in terms of the cohesiveness which the key players maintain in the network, thus it measures the reduction in cohesiveness of the network in the absence of the key players. It is easy to see that the KPP-Neg is closely related to the critical node detection problem since both problems seek nodes of the network whose absence (removal) would compromise the cohesion of the network. Infact, the metrics proposed for the KPP-Neg in Borgatti (2006) appear as connectivity objective functions in some of the mathematical programming models developed for both fragmentation-based CNDP (e.g CNP, MinMaxC, CC-CNP) and the distance-based CNDP (e.g Harary index, Wiener index). Thus, the models

which were developed for the critical node detection problem can be applied to the key player problem/Negative in social networks.

The second key player problem KPP-Pos examines the extent to which key players are connected to and embedded in the network thus they are nodes which are maximally connected to the rest of the nodes in the network. The KPP-Pos is useful in applications requiring quick diffusion of information or practices to members of a social network using a few individuals as seeds. The goal of the KPP-Pos can be seen as the opposite of that of the CNDP. Nevertheless, as the KPP-Pos is inherently combinatorial and deal with speed of diffusion, the distance-based CNP variant would be the closest related CNDP variant to the KKP-Pos.

Maximum s-club problem

Another class of graph problems that is closely related to the critical node detection problem is the s-club problem (Mokken et al. 1979). Given an unweighted graph $G = (V, E)$, an s-club is defined as a subset $S \subset V$ of nodes whose induced subgraph $G[S]$ has a diameter at most s. The maximum s-club problem therefore is to identify an s-club of maximum cardinality, that is, to find a maximum cardinality subgraph with pairwise distance at most s. The maximum s-club problem is one of the most useful distance-based clique relaxation models for identifying cohesive subgroups in social networks (Shahinpour & Butenko 2013). In Veremyev et al. (2015), the authors showed adaptation of their compact model for the distance-based critical node detection problem to the s-club problem indicating the potential for adapting the models proposed in this thesis to the maximum s-club problem.

Survivable network design problem

The devastating effects and frequency of natural disasters on communication networks motivated the problem of survivable network design (Grötschel et al. 1995). A network is said to be survivable if it can establish communications between its nodes even after failures of a pre-defined number of nodes or links while vulnerability relates to changes

in the distance between pairs of nodes as a result of edge or node removals (Gouveia & Leitner 2017). The survivable network design problem is the problem of designing minimum cost networks that satisfy certain connectivity requirements (Grötschel et al. 1995). In earlier studies, the connectivity requirements which represent network survivability were modelled using well-known graph problems such as Steiner tree and minimum cost $k-$connected network problems. Due to the importance of quality of service, recent studies in survivable network design combine survivability and quality of service by imposing hop-constraints (see e.g Gouveia et al. (2006), Mahjoub et al. (2013), Gouveia & Leitner (2017)). One of such problems that relates to the critical node detection problem is the network design problem with vulnerability constraints (NDPVC) proposed by Gouveia & Leitner (2017). The NDPVC seeks a minimum cost subgraph in which for every commodity (s,t), there is a path of length at most H_{st} and after removal of $k-1$ edges, the resulting graph has a path of length at most H'_{st}. In the NDPVC, vulnerability is associated with changes in hop distance between node pairs after deletion of edges from the network. The NDPVC considers all possible edge deletions that is, the minimum cost subgraph has to be robust to any edge deletion making the concept of "critical" edge not important. Notwithstanding, it relates to the edge version of the distance-based critical node detection problem since the actual distances between pre-defined commodities (s,t) is accounted for as well as number of edges to be deleted.

2.5 Concluding remarks

In this chapter, a review of literature on node and edge deletion problems was presented. Beginning with the history of node and edge deletion problems, we reviewed problems that relate to a very important property in network which is connectivity. One of such problems is the network interdiction problem which come in different flavours including the minimum cost flow interdiction, shortest path interdiction etc. Each of these flavours of the network interdiction problem is constituted by an edge deletion problem at its defender's level. We then presented a detailed review of the problem of interest of

this thesis which is the critical node detection problem. Under the two broad groups of the critical node detection problem namely fragmentation-based and distance-based, different objective metrics were considered. Models and algorithms for different variants of the critical node detection problems were reviewed. The review showed that the fragmentation-based critical node detection problem has been well-solved both from an exact algorithm point of view as well as heuristic. However, very little amount of literature exists for the distance-based critical node detection problem especially from an algorithm perspective hence the need for this research. In the next chapter, we formally present the distance-based critical node detection problem as well new models and exact algorithm which we have developed to solve it.

Chapter 3

Exact methods for the distance-based critical node detection problem

3.1 Introduction

Assessment of system vulnerability to adversarial attacks has become an important concern to organisations, especially in the wake of security threats around the world. Natural occurrences such as environmental disasters and disease epidemics with their cascading effects impact the overall performance of systems, in which they occur. Networked system developers now consider as a matter of necessity the issue of robustness at the very stage of building new networks. For instance, they try to ensure the existence of many unique alternative routes or multiple connectivity between specified source and destination pairs as possible (Gouveia & Leitner 2017). Similarly, existing networked system owners continue to seek efficient surveillance strategies to ensure minimal disruption in the health and performance of their system. An important strategy for the latter case is to identify elements of a given system that are critical in maintaining optimum system performance. In other words, we are identifying the parts of a system whose

failure would result in a break down of the system. System performance has varying definitions depending on the topology of the system and the application being considered. Nevertheless, the underlying problem is that of identifying important system elements. The associated problem as studied in the optimization community is termed critical node detection problem (CNDP), as introduced in Arulselvan et al. (2009). Given an undirected graph $G = (V, E)$ with $n = |V|$ nodes (vertices) and $m = |E|$ edges (arcs), the problem is to identify a subset of nodes of limited cardinality whose deletion results in a subgraph of maximum disconnectivity with respect to a predefined connectivity metric.

An essential aspect in the study of the critical node detection problem is the identification of a network property relevant to the network under study and an appropriate metric for its description (Borgatti & Everett 2006). This is primarily determined by the application context. In relation to network properties associated with studies on the CNDP, existing studies can be grouped into two broad categories, namely fragmentation-based and distance-based connectivity. A significant volume of research has been carried out in relation to the former category as shown in the review chapter (see Chapter 2). However, only a few computational studies have been done with regards to the latter which is the distance-based critical node detection problem (DCNDP).

In this chapter, we focus on the distance-based critical node detection problem whose goal is to minimise some distance-based connectivity objective by the deletion of a subset of nodes of specified cardinality. We begin by presenting the different distance-based connectivity functions as defined in Veremyev et al. (2015) as well as the integer programming formulation proposed by the authors. The IP model has $O(|V|^3)$ variables and $O(|V|^2|E|)$ constraints and is sensitive to edge-dense graphs as well as graphs with short average path lengths. We use this model as a base model to benchmark our study. We then present our new formulations, the first is a generic integer programming model which like the base model is a valid formulation for all the distance-based connectivity functions defined in Veremyev et al. (2015). For a specific distance-based connectivity function, we also propose a reduced formulation that has fewer variables compared to the first model. We exploit the structure of the problems

to develop an efficient separation routine to use within a branch-and-cut framework. Our algorithmic framework for solving the proposed models is based on a modified breadth-first-search tree generation. We demonstrate the efficiency of our approach in comparison to the base model on both real-world and synthetic graphs using two classes of the distance-based critical node detection problem. These two classes have interesting practical applications and broadly generalise the other classes. Comparing our implementation for the two distance DCNDP classes on some real-world graphs provide computational evidence to the impact of connectivity objective on the possibility and ease of solving the CNDP. We also propose valid inequalities based on mixed integer rounding. Our contributions are two-fold:

i From the methodological side, we introduce new formulations that use a decomposition approach. It was designed to exploit certain classes of distance functions (when L is small) as the natural compact model can become too big. We also introduce two families of valid inequalities for these problems.

ii We perform a computational study to test and compare the performance of the new formulations we introduced on a number of real-world and computer generated instances. The implementation includes a) a modified breadth-first-search (BFS) algorithm to separate both integer and fractional solutions b) a primal heuristic to improve upper bounds c) a family of valid inequalities to improve lower bounds.

Organisation

The rest of the chapter is organised as follows. In Section 3.2, we formally describe the distance-based critical node detection problem and give definition of the distance-based connectivity metrics proposed by Veremyev et al. (2015). We also present the base integer programming formulations proposed by the authors. In section 3.3, we present our new path-based mixed integer programming formulations. In section 3.4, we present our modified breadth-first-search algorithm for solving the separation problem associated

with the new models as well as valid inequalities and primal heuristics. We present our computational study with thorough comparisons of models in sections 3.5 and 3.6. Concluding remarks are provided in section 3.7.

3.2 Problem description and base formulation

Let $G = (V, E)$ be a simple undirected unweighted graph with a finite set V of nodes (or vertices) and a finite set $E \subseteq V \times V$ of edges where $n := |V|$ and $m := |E|$. For any given node say i, denote by $\mathcal{N}_G(i)$, the set of all neighbours of i that is, $\mathcal{N}_G(i) = \{j \in V : (i, j) \in E\}$. Node i is said to be connected to another node j if there is a path between them. A shortest path between i and j is a path containing the least number of edges among all paths connecting i and j. The number of edges of a shortest path between i and j in G is known as the hop distance or simply distance between i and j denoted by $d_G(i, j)$. The maximum distance between any pair of nodes in G is known as the diameter of G. The goal of the distance-based critical node detection problem is to find a subset of nodes $R \subset V$ with $|R| \leq B$, whose removal minimises some distance-based connectivity measure $f(d)$ written mathematically as:

$$\textbf{DCNDP} : \min_{R \subset V} \sum_{i,j \in V : i<j} f(d_{G^R}(ij)) : |R| \leq B$$

where $d_{G^R}(ij)$ is the distance or length of shortest path between nodes i and j in the node-deleted subgraph $G^R = G[V \setminus R]$, $f : Z_{>0} \cup \{\infty\} \rightarrow R$ represents some distance-based connectivity function and B is the available budget on the set of critical nodes. The distance connectivity function $f(.)$ is assumed to be non-increasing which implies that paths with larger lengths contribute less in the objective function. Also, for any pair of node $i, j \in V \setminus R$, if i and j are not connected, then $d_{G^R}(ij) = \infty$ and we assume that $f(\infty) = 0$. For simplicity of notation, we use $f(d)$ instead of $f(d_{G^R}(ij))$ hereafter.

3.2.1 Classes of the distance-based critical node detection problem

The concept of the connectivity metrics used in the definition of the distance-based critical node detection problem come from notions of cohesion and fragmentation in network analysis. For example, the characteristic path length and efficiency (Zhou et al. 2008) defined respectively by:

$$\mathcal{L}(G) = \frac{1}{n(n-1)} \sum_{i,j \in V: i \neq j} d_G(i,j) \quad \text{and} \quad \mathcal{E}(G) = \frac{1}{n(n-1)} \sum_{i,j \in V: i \neq j} \frac{1}{d_G(i,j)}$$

are popular distance measures used in the social network analysis literature. In Veremyev et al. (2015), five different classes of the DCNDP were introduced. Each class corresponds to a different distance connectivity function $f(.)$. However the underlying idea in each of the functions is that the set of critical nodes realised by solving the associated optimisation problem ensures that pairwise distances in the induced subgraph are maximised.

Class 1. *Minimise the number of node pairs connected by a path of length at most k:*

$$f(d) = \begin{cases} 1, & \text{if } d \leq k \\ 0, & \text{if } d > k \end{cases} \tag{3.1}$$

where k is a given positive integer representing the cut-off hop distance. The special case where $k \geq n-1$ is the classical CNP version which minimises the number of connected node pairs. Interesting instances of this class of DCNDP would be graphs with a small diameter and thus a large proportion of nodes connected within a small number of hops. We will refer to the DCNDP version corresponding to this class of distance function as DCNDP-1.

Class 2. *Minimise the Harary index or efficiency of the graph*

$$f(d) = \begin{cases} d^{-1}, & \text{if } d < \infty \\ 0, & \text{if } d = \infty \end{cases}$$

This metric is based on the assumption in communication network analysis that communication efficiency between node pairs is inversely proportional to the distance between them

(Crucitti et al. 2003). Typically, two disconnected nodes are at a distance of ∞. However, in many real-world applications such as telecommunication and social networks, two nodes that are far enough from each other cannot effectively communicate. Hence, we consider a threshold model, where two nodes separated by a distance that is more than some specified threshold L, cannot communicate. This results in the following modified Harary distance function.

$$f(d) = \begin{cases} d^{-1}, & \text{if } d \leq L \\ 0, & \text{otherwise} \end{cases} \tag{3.2}$$

We will refer to the DCNDP version corresponding to this connectivity function as DCNDP-2.

Class 3. *Minimise the sum of power functions of distances in the graph:*

$$f(d) = \begin{cases} p^d, & \text{if } d < \infty \\ 0, & \text{if } d = \infty \end{cases} \tag{3.3}$$

where p is a fixed parameter such that $0 < p < 1$. The distance function which is also known as the Hosoya polynomial (Hosoya 1988) in chemistry can be applied in the context of virus or rumour propagation. The parameter p here models the transmission probability of the phenomenon between two adjacent nodes. For non-adjacent node pairs, the transmission probability estimated from the power function would decrease with increase in the distance (d) between the nodes. Hence, class 3 of the DCNDP which we refer to as DCNDP-3 can be understood as identifying a subset of nodes whose removal maximally reduces the network's propensity to spread a virus.

Class 4. *Maximise the generalised Wiener index or, equivalently, the characteristic path length of the graph:*

$$f(d) = \begin{cases} -d, & \text{if } d < \infty \\ -M, & \text{if } d = \infty \end{cases} \tag{3.4}$$

where M is a sufficiently large constant such that $M > n - 1$.

The distance function $f(.)$ is given here as negative since the DCNDP has been defined as a minimisation problem. One can easily reformulate as a maximisation problem that maximises the sum of all pairs shortest path distances in the induced subgraph as in Hooshmand et al. (2020).

Class 5. *Maximise the shortest path length between nodes s and t in the graph:*

$$f(d) = \begin{cases} 0, & \text{if } i \neq s, \text{ or } j \neq t \\ -M, & \text{if } i = s, j = t, \text{ and } d = \infty \\ -d, & \text{otherwise} \end{cases} \tag{3.5}$$

where $s, t \in V \setminus R$.

This is simply the node deletion version of the defender's problem in the classical shortest path network interdiction problem (Israeli & Wood 2002) which maximises the distance between node (commodity) pair (s, t). This can also be modified to handle non-singleton set of commodity pairs.

3.2.2 Base MIP formulations

Given the input graph $G = (V, E)$ as defined before with $n = |V|$ nodes and $m = |E|$ edges, also let B and L be given positive integers. Associated with any node $i \in V$, we define a variable x_i as

$$x_i = \begin{cases} 1, & \text{if node } i \text{ is deleted,} \\ 0, & \text{otherwise.} \end{cases} \tag{3.6}$$

Similarly, we define connectivity variables by

$$y_{ij}^{\ell} = \begin{cases} 1, & \text{if } (i, j) \text{ are connected by a path of length } \leq \ell \text{ in } G^R \\ 0, & \text{otherwise.} \end{cases} \tag{3.7}$$

The generic integer programming formulation provided in Veremyev et al. (2015) is given as follows:

DCNDP-base:

$$\text{Minimise} \quad \sum_{i,j\in V:i<j}\left(f(1)y_{ij}^{1}+\sum_{\ell=2}^{L}f(\ell)\left(y_{ij}^{\ell}-y_{ij}^{\ell-1}\right)\right) \tag{3.8}$$

$$+\sum_{i,j\in V:i<j}f(\infty)\left(1-y_{ij}^{L}\right) \tag{3.9}$$

$$\text{s.t.} \quad y_{ij}^{1}+x_i+x_j\geq 1, \quad \forall(i,j)\in E,\ i<j \tag{3.10}$$

$$y_{ij}^{\ell}=y_{ij}^{1}, \quad \forall(i,j)\in E,\ i<j,\ \ell\in\{2,\ldots,L\} \tag{3.11}$$

$$y_{ij}^{\ell}+x_i\leq 1, \quad \forall(i,j)\in V,\ i<j,\ \ell\in\{1,\,2,\ldots,L\} \tag{3.12}$$

$$y_{ij}^{\ell}+x_j\leq 1, \quad \forall(i,j)\in V,\ i<j,\ \ell\in\{1,\,2,\ldots,L\} \tag{3.13}$$

$$y_{ij}^{\ell}\leq\sum_{t:(i,t)\in E}y_{tj}^{\ell-1}, \quad \forall(i,j)\notin E,\ i<j,\ \ell\in\{2,\ldots,L\} \tag{3.14}$$

$$y_{ij}^{\ell}\geq y_{tj}^{\ell-1}-x_i, \quad \forall(i,t)\in E,\ (i,j)\notin E,\ i<j,\ \ell\in\{2,\ldots,L\} \tag{3.15}$$

$$\sum_{i\in V}x_i\leq B \tag{3.16}$$

$$x_i\in\{0,\,1\}, \quad \forall i\in V \tag{3.17}$$

$$y_{ij}^{\ell}\in\{0,\,1\}, \quad \forall(i,j)\in V,\ i<j,\ \ell\in\{1,\ldots,L\} \tag{3.18}$$

Objective terms (3.8)–(3.9) minimise some distance-based connectivity objective $f(.)$. In objective (3.8), each term of the form $(y_{ij}^{\ell}-y_{ij}^{\ell-1}), \ell\geq 2$ is equal to 1 iff $y_{ij}^{\ell}=1$ and $y_{ij}^{\ell-1}=0$. This implies that ℓ is the shortest path length between i and j in G^R. Moreover, the term $1-y_{ij}^{L}$ in objective (3.9) is equal to 1 iff nodes i and j are disconnected. Constraints (3.10)–(3.11) ensure that $y_{ij}^{\ell}=1$ for adjacent node pairs (i,j), if neither of the nodes i nor j is deleted. Constraints (3.12)-(3.13) enforce y_{ij}^{ℓ} to be zero if either node i or j is deleted. Constraints (3.14)-(3.15) ensure that there is a path of length at most ℓ between non-adjacent nodes i and j iff node i is not deleted and there is a path of length at most $\ell-1$ between t and j for some non-deleted node t in the neighbourhood of node i. Constraint (3.16) limits the cardinality of the critical node set to the budget

B. Constraints (3.17)-(3.18) are binary restrictions on the decision variables. The observation that the binary restrictions on the y variables can be relaxed was made in Veremyev et al. (2015). Intuitively, once the x variables are fixed, constraints (3.10) and (3.15) will fix a subset of y variables to either 0 or 1. The minimisation function incentivises the remaining y variables to take a value of 0.

As noted in Veremyev et al. (2015), considering initial shortest-paths d_{ij} between each node pair (i, j) in the input graph enables us to set connectivity variables $y_{ij}^{\ell} = 0$ for all $\ell < d_{ij}$ as well as defining constraints (3.14)-(3.15) for only $\ell \geq d_{ij}$. In addition to this, we note that only neighbours t of i having shortest path $d_{tj} \leq \ell - 1$ should be considered in constraints (3.14)-(3.15). These and leaf-node based considerations reduce the number of variables and constraints in the model, thereby improving performance of standard IP solvers. We refer to this base formulation with all the proposed enhancements as the enhanced compact model (ECM).

3.3 New integer programming formulations

The motivation behind the distance-based connectivity metrics is that shorter paths contribute more to the connectivity objective than longer paths. Since only paths of length at most L is of importance here, by keeping track of paths within this threshold, we can guarantee that any given node pair (i, j) is L-distance disconnected iff at least one node along all candidate paths $\mathcal{P}_L(i, j)$ between i and j is deleted. We use this idea to formulate new integer programming models which are based on the paths connecting node pairs and as such consists of exponentially many constraints. However, only few of these constraints would eventually be active in the formulation hence they can be treated as lazy constraints to be separated in polynomial time by solving a shortest path problem.

Using the decision variables x and y as defined in 3.6 & 3.7, we propose a new path-based model which like the base model admits all the 5 distance-based connectivity functions. We refer to this model as **DCNDP-PBML** and the mixed integer formulation is given

as follows:

DCNDP-PBML

$$\text{Minimise} \quad \sum_{i,j \in V: i<j} \left(f(1) y_{ij}^1 + \sum_{\ell=2}^{L} f(\ell) \left(y_{ij}^{\ell} - y_{ij}^{\ell-1} \right) \right) \tag{3.19}$$

$$+ \sum_{i,j \in V: i<j} f(\infty) \left(1 - y_{ij}^{L} \right) \tag{3.20}$$

$$\text{s.t.} \quad \sum_{r \in V(P)} x_r + y_{ij}^{|P|} \geq 1, \quad \forall P \in \mathcal{P}_L(i,j),\ i,j \in V,\ i<j \tag{3.21}$$

$$y_{ij}^{\ell-1} \leq y_{ij}^{\ell}, \quad \forall (i,j) \in V,\ i<j,\ \ell \in \{2,\ldots,L\} \tag{3.22}$$

$$y_{ij}^{\ell} = y_{ij}^{1}, \quad \forall (i,j) \in E,\ i<j,\ \ell \in \{2,\ldots,L\} \tag{3.23}$$

$$\sum_{i \in V} x_i \leq B \tag{3.24}$$

$$x_i \in \{0,\, 1\}, \quad \forall i \in V \tag{3.25}$$

$$y_{ij}^{\ell} \in \{0,\, 1\}, \quad \forall (i,j) \in V,\ i<j,\ \ell \in \{1,\ldots,L\} \tag{3.26}$$

The objective terms (3.19)–(3.20) are the same with those of **DCNDP-base**. Constraints (3.21) ensures that node pairs (i,j) are L-distance disconnected iff at least one node along all paths of length less or equal to L connecting i and j is deleted. Since there are potentially many such constraints, some of which are redundant, we explicitly model the non-redundant constraints ($y_{ij}^{\ell} + x_i + x_j \geq 1$) for edges and leave the rest to be identified in a separation routine. Constraints (3.22) ensure that there is a path of length l, if there is a path of length $l-1$. Constraints (3.23) is same as constraints (3.11). The budgetary constraint limiting the cardinality of the critical node set is represented by constraint (3.24) while constraints (3.25)–(3.26) specify the domains of the decision variables. Following the same argument as in Veremyev et al. (2015), the integrality constraints can also be relaxed for the connectivity variables y_{ij}^{l} in this formulation. Note that the root LP-relaxation of the base model (formulation (3.8)–(3.18)) will be better than that of the path-based model (formulation (3.19)–(3.26)) as constraints (3.10) and (3.15) of the base model would completely imply constraints (3.21). However, constraints (3.14) and (3.15) make the compact model grow with the graph size.

Furthermore, we exploit the nature of the distance-based connectivity function (3.1) of class 1 of the DCNDP by observing that the objective function does not explicitly make use of the path lengths indexed by ℓ in the y variables. Instead, it counts whether or not pairs of nodes (i, j) are connected by a path of length ℓ. This enables us to eliminate the ℓ- index in the distance connectivity variable y_{ij}^{ℓ} used in the path-based model (formulation (3.19)–(3.26)) to arrive at an alternative path model for DCNDP-1 which we denote by **DCNDP-1-PBM**. We can get computationally more efficient solutions using **DCNDP-1-PBM** as the corresponding LP relaxation is relatively smaller, which we exploit to compute quicker bounds. These observations are supported in our experiments (see Sections 3.6.1 & 5.2.1).

We define a new set of connectivity variables y_{ij} as follows:

$$y_{ij} = \begin{cases} 1, & \text{if } (i,j) \text{ are connected by a path of length} \leq k \text{ in } G^R \\ 0, & \text{otherwise.} \end{cases} \tag{3.27}$$

Using the former set of node deletion variables (3.6) along with the new connectivity variables (3.27), the reduced path-based model for DCNDP-1 is formulated as follows:

DCNDP-1-PBM

$$\text{Minimise} \quad \sum_{i,j \in V: i<j} y_{ij} \tag{3.28}$$

$$\text{s.t.} \quad \sum_{r \in V(P)} x_r + y_{ij} \geq 1, \quad \forall P \in \mathcal{P}_k(i,j),\ (i,j) \in V,\ i<j \tag{3.29}$$

$$\sum_{i \in V} x_i \leq B \tag{3.30}$$

$$x_i \in \{0,\ 1\}, \quad \forall i \in V \tag{3.31}$$

$$y_{ij} \in \{0,\ 1\}, \quad \forall (i,j) \in V,\ i<j \tag{3.32}$$

Objective function (3.28) minimises the number of node pairs connected by a path of hop distance at most k. The ideas behind constraints (3.29)- (3.32) are similar to those of constraints (3.21); (3.24)–(3.26) of the extended path model (formulation (3.19)–(3.26)).

We can relate both path-based formulations in the following way. Let

$$\mathcal{Q} := \{(\mathbf{y}, \mathbf{x}) : (\mathbf{y}, \mathbf{x}) \text{ satisfies constraints (3.21) to (3.26)}\}$$

where $\mathbf{y} = \mathbf{y}^L, \ldots, \mathbf{y}^1$. Then we can think of the feasible space of the reduced path formulation **DCNDP-1-PBM** as

$$Proj_{(\mathbf{y}^L, \mathbf{x})}\mathcal{Q} := \{(\mathbf{y^L}, \mathbf{x}) : \exists(\mathbf{y}^{L-1}, \ldots, \mathbf{y}^1) : (\mathbf{y}, \mathbf{x}) \in \mathcal{Q}\}$$

If we do Fourier elimination (Williams 1986) of variables systematically starting from y_{ij}^1 all the way to y_{ij}^k, we will get exactly **DCNDP-1-PBM** with no changes. On the other hand, if we interpret y_{ij} as non-negative continuous variables that define the value of the distance function between nodes i and j, then we can model **DCNDP-PBML** similar to the reduced version **DCNDP-1-PBM**. By removing the dependency on ℓ in y variables, constraint (3.21) can now be replaced by

$$\sum_{r \in V(P)} f(|P|)x_r + y_{ij} \geq f(|P|) \quad \forall P \in \mathcal{P}_L(i,j),\ i,j \in V,\ i < j \tag{3.33}$$

Objective function (3.19)–(3.20) would be replaced by

$$\min \sum_{i,j \in V : i<j} y_{ij}$$

we can then drop constraints (3.22)–(3.23) and relax constraint (3.26) to $y_{ij} \in [0,1]$. By replacing y_{ij}^l with the aggregated continuous y_{ij} variables, we can see that the generic path formulation (3.19)–(3.26) can be reformulated to a form of the reduced path formulation (3.28)–(3.32) accommodating DCNDP-2 and DCNDP-3 objectives. This reformulation can also accommodate positive integer edge weights and easily adapted to the remaining distance classes. We discuss these in detail in Chapter 5.

3.4 Solution Methods

In this section, we present details of the separation routine for the path-based formulations. We also present a heuristic framework for generation of good incumbent solution as well as valid inequalities for improvement of lower bounds.

3.4.1 Separation algorithm

Instead of solving a hop-constrained shortest path problem (Guérin & Orda 2002) or enumerating all paths of certain length for every pair of nodes in order to generate cuts, we use a customised approach to separate violated lazy cuts. This approach involves in generating a breadth-first-search tree for every node $v \in V$. At each level i of the BFS tree rooted at v, there are k_i nodes $\{l_1^i, \ldots, l_{k_i}^i\}$ that are at a distance of i from v (see Figure 3.1). The unique path from a node in the BFS tree to the root node gives us an inequality of type (3.21) or (3.29). For model **DCNDP-1-PBM**, since we are only interested in paths up to a specific length k, we stop the traversal up to that particular depth. For **DCNP-PBML**, we continue traversal up to depth L. In the BFS tree, we identify the path from the root node i to the nodes in level $\ell = 1, \ldots, L$ and if this path is violated, we add it as a cut. The generation of these BFS trees is much more efficient and we get multiple paths for one such tree. BFS tree can be generated in $O(|E|)$ as compared to solving shortest path that will take $O(|E| + |V| \log |V|)$. Since we only explore L levels of the BFS tree, it takes far less time than $O(|E|)$ to generate these trees. A pseudocode of the algorithm is given by Algorithm 1.

The algorithm begins by selecting a root node r from the set of candidate root nodes and constructs a BFS tree up to a specified depth (or tree level) L. The algorithm follows a standard BFS algorithm with the only difference being that it keeps track of the depth of each discovered node (lines 6 & 10) with which it determines when to terminate the current tree generation and also to separate the corresponding path inequalities. For each node t that is being visited, the algorithm proceeds in one of three possible ways depending on its tree level $l[t]$.

Case 1: $l[t] = 1$ which implies that node t is a direct neighbour of the root node r that is $(r, t) \in E$. In this case, we only add t to the queue Q and mark it as visited (lines 11-13) since inequalities for edges are already present in the formulation.

Case 2: $1 < l[t] \leq L$, which implies that $(r, t) \notin E$ but hop distance between root node r and t is less than or equal to L. This is the case of interest. Hence we not only add t to the queue Q and mark it as visited, but we also check if path inequalities (3.21)

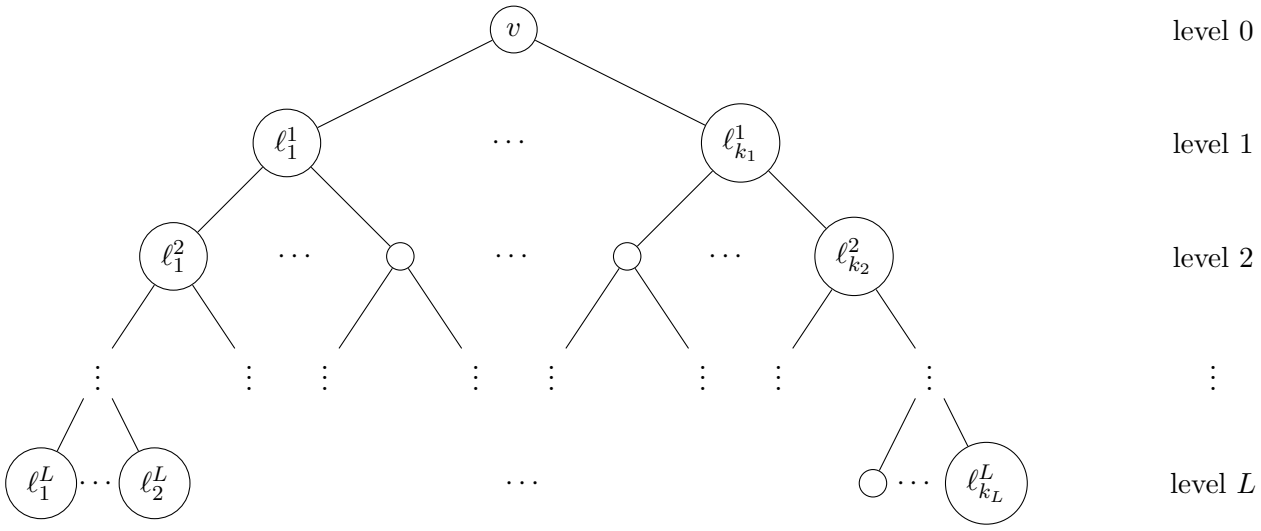


Figure 3.1: An example of an L-depth BFS tree generation where L is the depth of the tree and is based on the threshold hop distance specified in the corresponding DCNDP class. Branches of the tree need not be binary

or (3.29) of the corresponding shortest path P_{rt} are violated (lines 14-17) and add the violated inequality to the queue Q_c.

Case 3: $l[t] > L$, this implies that all nodes reachable within hop distance L from the root r have been explored. Therefore, we stop the current tree generation (lines 18-20) and return to line 3 where we select the next root and begin the process all over.

We note a slight difference in the implementation of Algorithm 1 for separation of integer and fractional solutions. Observe that in any incumbent solution, the variables are either 0 or 1. Since constraints (3.21) or (3.29) are only violated when left-hand sum is less than 1, only nodes with value $\tilde{x}_i = 0$ in any given incumbent solution would potentially lead to violated path constraints. Therefore, the set of candidate root nodes R_c in step 3 of Algorithm 1 consists only of nodes with value $\tilde{x}_i = 0$ in the current integer solution. Following this thought, in exploring a node in step 9, we also limit it to unvisited neighbors whose value in the current incumbent solution is zero. Hence, only nodes with zero solution value ($\tilde{x}_i = 0$) feature in the constructed trees. This helps us to detect and add all violated constraints for any given integer solution and avoids spending time in unpromising branches.

The separation problem for the most violated cut of a fractional solution involves solving a shortest path problem with edge weights in an auxiliary graph. The edge weights are the LP relaxation values ($\bar{x}$). This can still be done in polynomial time through a transformation to a directed graph as the weights are positive. This, however, increases the graph size and the shortest path problem has to be solved for all pairs of nodes. We instead adapt our BFS algorithm as a heuristic to separate a violated fractional solution. The BFS trees are built based on the LP relaxation values $\bar{x}_i$, i.e., candidate root nodes with smaller LP relaxation values are chosen first for BFS tree generation. Also, nodes are explored in increasing order of their neighbours' LP-relaxation values, this means that the unvisited neighbours with smaller LP relaxation values $\bar{x}_i$ are visited first before other neighbours. This ensures that the most violated constraints for all paths of a particular hop distance between node pairs are separated. Furthermore, for LP relaxation solutions, we set limits on the number of cuts added at which we end

Algorithm 1: Separation algorithm for the proposed model

Input : Graph $G = (V, E)$, Incumbent $(\tilde{x}, \tilde{y})$ or LP-relaxation solution $(\bar{x}, \bar{y})$, set of candidate root nodes R_c, and tree depth L

Output: Queue of violated inequalities Q_c

for $r \in R_c$ **do**

  $Q \leftarrow \{r\}$; `// initialise queue` Q `with root node` r

  $T \leftarrow \{r\}$; `// mark` r `as visited`

  $l[r] = 0$;

  while $Q \neq \emptyset$ **do**

    $s \leftarrow Q.remove$; `// retrieve the first element in queue` Q

    `/* Explore node` s`, where` δ_s `denotes neighbours of` s `*/`

    for $t \in \delta_s \setminus T$ **do**

      $l[t] = l[s] + 1$;

      if $l[t] = 1$ **then**

        $Q.add(t)$; `// add` t `to queue` Q

        $T \leftarrow T \cup \{t\}$; `// mark` t `as visited`

      else if $l[t] \in [2, L]$ **then**

        $Q.add(t)$;

        $T \leftarrow T \cup \{t\}$; `// mark` t `as visited`

        `/* check violation of` (3.29) `(resp.` (3.21) `for DCNDP-2b)`

        `for the path` P_{rt} `*/`

        if $\sum_{i \in V(P_{rt})} \tilde{x}_i + \tilde{y}_{rt} < 1$ *(resp.* $\sum_{i \in V(P_{rt})} \tilde{x}_i + \tilde{y}_{rt}^{l[t]} < 1$*)* **then**

          $Q_c.add(\sum_{i \in V(P_{rt})} x_i + y_{rt} \geq 1)$

          (resp. $Q_c.add(\sum_{i \in V(P_{rt})} x_i + y_{rt}^{l[t]} \geq 1)$);

        end

      else

        $Q \leftarrow \emptyset$;

        break;

      end

    end

  end

end

the call on the separation algorithm and re-solve the problem. We set parameter for cut limit to 150 cuts, after trying different values within the neighborhood of ± 100. For integer solutions, we only set this limit to stop the current BFS tree generation and return to line 3 to generate a new BFS tree rooted at the next candidate root node. Nevertheless, this cut limit is only applied after the optimisation process begins branching to ensure all violated cuts are separated. This is because at the root node of a branch-and-bound tree, many integer solutions are infeasible to the original problem, hence more constraints would be potentially violated.

3.4.2 Valid inequalities

In addition to the lazy cuts, we propose two families of strong valid inequalities that are not implied by the constraints in the formulations. These inequalities can be viewed as $\{0, \frac{1}{2}\}$ (or $\{0, \frac{2}{3}\}$)-Chvátal-Gomory (CG) cuts, where the constraints are multiplied by constants in the set $\{0, \frac{1}{2}\}$ (or $\{0, \frac{2}{3}\}$), added together and then rounded up. Note that this generalises the procedure that was provided in Di Summa et al. (2012) and we can obtain a broader class of inequalities. For the purpose of our computational experiments, we focused on odd cycles of lengths 3 and 5 as larger holes are atypical in small world networks.

Odd holes of length 3

Let H be an odd hole of length 3 in $G = (V, E)$ with node set $V(H)$ and edge set $E(H)$. Also for simplicity, let $V(H) = \{1, 2, 3\}$ and $E(H) = \{12, 23, 13\}$.

Proposition 3.4.1. *The following is a valid inequality for the path-based constraints in $E(H)$:*

$$x_1 + x_2 + x_3 + y_{12} + y_{13} + y_{23} \geq 2 \tag{3.34}$$

Proof. Consider the following valid inequalities for each edge in H:

$$x_1 + x_2 + y_{12} \geq 1$$

$$x_1 + x_3 + y_{13} \geq 1$$

$$x_2 + x_3 + y_{23} \geq 1$$

Summing together we get $2x_1+2x_2+2x_3+y_{12}+y_{13}+y_{23} \geq 3$. Then by applying $\{0, \frac{1}{2}\}$-CG procedure we obtain inequality (3.34). It is easy to show that inequality (3.34) is not implied by inequalities (3.29)-(3.32) of model **DCNDP-1-PBM**. The fractional point, $x_i = \frac{1}{2}$ for all $i \in V(H), y_{ij} = 0$ for all $i, j \in E(H)$, is feasible to the LP-relaxation of **DCNDP-1-PBM** but violates (3.34). □

Odd holes of length 5

Let H be an odd hole of length 5 in $G = (V, E)$ with node set $V(H)$ and edge set $E(H)$. Also for simplicity, let $V(H) = \{1, 2, 3, 4, 5\}$, $E(H) = \{12, 23, 34, 45, 15\}$ and let $P_2(i, j)$ denote the path of length 2 connecting i and j in H. Formulation (3.28)-(3.32) contains the following constraints:

path of length 1: $x_i + x_j + y_{ij} \geq 1 \quad i, j \in E(H)$

path of length 2: $x_i + x_j + x_k + y_{ij} \geq 1 \quad i, j \notin E(H), k \in V(P_2(i, j)) \setminus \{i, j\}$

Proposition 3.4.2. *The following is a valid inequality for path-based constraints for paths of length 2 in $E(H)$:*

$$2x_1 + 2x_2 + 2x_3 + 2x_4 + 2x_5 + y_{13} + y_{24} + y_{35} + y_{14} + y_{25} \geq 4 \tag{3.35}$$

Proof. Consider the following valid inequalities corresponding to paths of length 2 in H:

$$x_1 + x_2 + x_3 + y_{13} \geq 1$$

$$x_2 + x_3 + x_4 + y_{24} \geq 1$$

$$x_3 + x_4 + x_5 + y_{35} \geq 1$$

$$x_1 + x_5 + x_4 + y_{14} \geq 1$$

$$x_2 + x_1 + x_5 + y_{25} \geq 1$$

Summing together we get $3x_1 + 3x_2 + 3x_3 + 3x_4 + 3x_5 + y_{13} + y_{24} + y_{35} + y_{14} + y_{25} \geq 5$. Then applying $\{0, \frac{2}{3}\}$-CG procedure, we obtain inequality (3.35). Inequality (3.35) is not implied by inequalities (3.29)-(3.32) of **DCNDP-1-PBM**. The fractional point, $x_i = \frac{1}{3}$ for all $i \in V(H), y_{ij} = \frac{1}{3}$ for all $i, j \in E(H)$, $y_{ij} = 0$ for all $i, j \notin E(H)$, is feasible to LP-relaxation of **DCNDP-1-PBM** but violates (3.35). □

Following the same procedure, the corresponding odd hole inequalities for **DCNDP-PBML** are:

$$x_1 + x_2 + x_3 + y_{12}^{(1)} + y_{13}^{(1)} + y_{23}^{(1)} \geq 2 \tag{3.36}$$

and

$$2x_1 + 2x_2 + 2x_3 + 2x_4 + 2x_5 + y_{13}^{(2)} + y_{24}^{(2)} + y_{35}^{(2)} + y_{14}^{(2)} + y_{25}^{(2)} \geq 4 \tag{3.37}$$

Separation of the odd hole inequalities is based on simple enumeration whereby a pool of cycles of given length is generated based on the cycle enumeration scheme proposed in Liu & Wang (2006) and the corresponding odd hole inequalities are routinely checked for violations.

3.4.3 Primal heuristic

Generation of good incumbent solution helps in pruning branch-and-bound nodes. Thus we propose a primal heuristic which incorporates information from LP-relaxation solutions into centrality-based ranking to arrive at a good primal bound. First, we extend the budget requirement to $\hat{B} = 1.5B$ instead of B, then based on the degree centrality measure, we obtain the top $\hat{B}$ ranking nodes. Using the LP-relaxation values of those nodes as selection probability, we randomly select B distinct nodes. We fix the x variables of the selected nodes to 1 ($x_i = 1$) while the x variables for the rest of the nodes are fixed to zero.

3.5 Computational experiments

3.5.1 Hardware & Software

Our computational study was performed on an HP computer equipped with Windows 8.1 x64 operating system, an Intel Core i3-4030 processor(CPU 1.90 GHz) and RAM 8GB. The models and algorithms were written in Python 3.6 (Anaconda 5) using Gurobi 8.1.0 (Gurobi Optimization 2018) as optimisation suite. We use NetworkX (Hagberg et al. 2008) for random graph generation and manipulating of the graphs. All experiments were run with time limit of 3600 seconds.

Graph	**n**	**m**	**diam**	**% k-Conn**
Hi-tech	33	91	5	88.3
Karate	34	78	5	85.6
Mexican	35	117	4	98.0
Sawmill	36	62	8	63.0
Chesapeake	39	170	3	100.0
Dolphins	62	159	8	58.5
Lesmiserable	77	254	5	85.4
Santafe	118	200	12	32.9
Sanjuansur	75	155	7	48.7
Attiro	59	128	8	68.0
LindenStrasse	232	303	13	12.1
SmallWorld	233	994	4	95.2
NetScience	379	914	17	13.3
USAir97	332	2126	6	84.8

Table 3.1: Characteristics of real-world graph instances.

3.5.2 Test Instances

Our test instances comprise both real-world and randomly generated graphs. Real-world instances are a subset of networks from the Pajek and UCINET datasets (Batagelj & Mrvar 2006, *UCINET software datasets* n.d.). All instances are connected graphs or the largest connected component of the original graph if the original graph is disconnected.

- **Hi-tech**($|V| = 33, |E| = 91$): Friendship network of employees in a hi-tech firm (Batagelj & Mrvar 2006, *UCINET software datasets* n.d.).
- **Karate** ($|V| = 34, |E| = 78$): A social network of a karate club at a U.S University in the 1970s (Batagelj & Mrvar 2006, *UCINET software datasets* n.d., Zachary 1977).
- **Mexican** ($|V| = 35, |E| = 117$) A network of relations (family, political and business) of political elite in Mexico (Batagelj & Mrvar 2006, *UCINET software datasets* n.d.).
- **Sawmill** ($|V| = 36, |E| = 62$): Communication network of employees within a small enterprise (Batagelj & Mrvar 2006, *UCINET software datasets* n.d.).
- **Chesapeake**($|V| = 39, |E| = 170$): Chesapeake Bay Mesohaline network (Batagelj & Mrvar 2006, Baird & Ulanowicz 1989).
- **Dolphins** ($|V| = 62, |E| = 159$): A social network that represents frequent associations between dolphins in a community in New Zealand (Batagelj & Mrvar 2006, *UCINET software datasets* n.d., Lusseau et al. 2003).
- **Lesmiserable** ($|V| = 77, |E| = 254$): Network of co-appearance of characters in the novel Les Miserable (Knuth 1993)
- **Santafe** ($|V| = 118, |E| = 200$): Collaboration network of scientists at the Santa Fe Institute (Girvan & Newman 2002).
- **SmallWorld** ($|V| = 233, |E| = 994$): A citation network (Batagelj & Mrvar 2006).

- **Sanjuansur** ($|V| = 75, |E| = 155$), and **attiro** ($|V| = 59, |E| = 128$) : Social networks of families in a rural area in Costa Rica (Batagelj & Mrvar 2006, *UCINET software datasets* n.d.).

- **LindenStrasse** ($|V| = 232, |E| = 303$): Network of friendly relationships between characters of the soap opera "Lindenstrasse" (Batagelj & Mrvar 2006).

- **USAir97** ($|V| = 332, |E| = 2126$): Transportation network of US airlines (Batagelj & Mrvar 2006).

- **NetScience** ($|V| = 379, |E| = 914$): Co-authorship network of scientists in science (*UCINET software datasets* n.d.).

The properties of the real-world instances are summarised on Table 3.1. We also used 3 classes of random graphs which were generated using the networkX random graph generators. For any graph in a particular class, 10 different instances were generated and results averaged and compared across those instances. The three classes are described as follows:

I **Barabasi-Albert random graphs:** The Barabasi-Albert model (Barabási & Albert 1999) is known for its preferential attachment mechanism wherein nodes with high degree have a higher propensity to be connected to a new node as the graph is grown. The degree distribution, defined to be the fraction of nodes with degree k, of the Barabasi-Albert model is known to follow a power law distribution $p_k \approx k^{-3}$. Instances of this random graph class were generated using networkX random graph generator with parameters $n = |V|$ and p, which denote the graph size and the number of edges to attach from a new node to existing nodes respectively. For the Barabasi-Albert graph class, two sets of graphs were generated namely, **ba1** and **ba2**, both with parameter $n = 100$ but with $p = 5$ and $p = 10$ respectively.

II **Erdos-Renyi random graphs:** Erdos-Renyi model (Erdös & Rényi 1959) for random graph generation defines a set of graphs having the same parameters $n =$

$|V|$, the size of the graph and p, probability of adding an edge between any two node pairs. Starting with an empty graph with $|V|$ nodes, the model creates a random graph by adding an edge between every pair of nodes $i, j \in V$ with probability p. The degree distribution of Erdos-Renyi graphs follow the Poisson distribution. Instances were generated using networkX random graph generator with parameter n denoting the graph size and p denoting the probability of edge creation. Two sets of graphs were generated using the Erdos-Renyi model. The first which is named **er1** has parameters $n = 80$ and $p = 0.15$ while the second named **er2** has parameters $n = 200$ and $p = 0.05$.

III **Uniform random graphs:** Given two input parameters n and m, the uniform random graph model $G_{n,m}$ returns a graph selected uniformly at random from set of all graphs having n nodes and m edges. Three sets of instances were generated using networkX random graph generator that takes n and m as parameters. The sets of instances are named **gnm1**, **gnm2** and **gnm3**.

The properties of the random graphs averaged over the 10 instances generated for each set are summarised in Table 3.2.

Graph	**n**	**m**	**diam**	**density (%)**	**k-DistConn (%)**	**efficiency(%)**
ba1	100	475	4.0	9.6	99.9	49.6
ba2	100	900	3.0	18.0	100	58.4
er1	80	470	3.0	14.9	100	55
er2	200	1004	4.0	5.0	97.7	42.8
gnm1	200	1000	4.2	5.0	97.9	42.8
gnm2	300	1500	4.4	3.3	94.1	39.6
gnm3	300	2000	4.0	4.5	99.6	43.4

Table 3.2: Characteristics of each set of random graph instances

3.5.3 Parameter settings and preprocessing

Our computational experiments comparing the base model and the new path-based models are based primarily on the first two classes (DCNDP-1 and DCNDP-2) of the distance-based critical node detection problem. For DCNDP-1, where we minimise the number of node pairs connected by a path of length at most k, we set the threshold distance $k = 3$. For DCNDP-2 which minimises the Harary index or efficiency, we set L to the diameter of the input graph. For experiments on randomly generated graphs, we set budget B to 5% and 10% of graph size. We varied this percentage for the real-world graphs from 1% to 10% of number of nodes in the input graph. The budget values for different instances are specified on the corresponding tables and figures in Section 3.6. We also explored the enhancements discussed in Section 3.2.2 as a preprocessing stage prior to running the Gurobi optimiser while fixing values for nodes with degree one to zero ($x_i = 0$) for both models.

3.6 Results and Discussion

The computational experiments were performed for varying sizes (nodes, edges) and classes of graphs with different edge densities and diameters. We compare the performance of the base model implemented with the suggested enhancements labelled as ECM and our path-based models labeled as PBM (or PBML for DCNDP-2). In each table, along with graph characteristics such as number of nodes, edges and diameter, we present computational times in seconds and/or percentage gaps for both the ECM and PBM for different budget settings. Columns labelled *InitObj* represent the initial objective values in the input graph prior to solving the model. For DCNDP-1, this is the percentage of node pairs connected by paths of length at most k ($k-$DistConn) while for DCNDP-2, it is the initial communication efficiency of the input graph. Similarly, columns labelled *FinObj* represent the final objective value (in percentage) at the end of optimisation or the best objective realised within the specified time limit. Columns labelled *ECMt* and *PBMt* represent the computational time for the base and path-based models respectively.

Recall that all experiments were run with time limit of 3600 seconds, hence, where the optimisation process could not terminate within the given time limit, the corresponding entry is marked $' > 3600'$. Similarly, when a problem instance runs out of memory, we indicate this by an "M". For instances which were unsolved within the specified time limit, the percentage gaps are calculated as

$$\%gap = 100 \cdot \frac{BestObj - FinalLowerBound}{FinalLowerBound}\%$$

For the synthetic graphs, the percentage gaps are averaged over all ten instances generated for each graph class.

3.6.1 Results of the base and the path formulations for DCNDP-1

The first set of experiments provides comparison between the base model (ECM) and the aggregated path-based model (PBM) based on DCNDP-1 that is, distance function 3.1.

					B=0.05n			**B=0.1n**		
Graph	**n**	**m**	**diam**	**InitObj**	**FinObj**	**ECMt (s)**	**PBMt (s)**	**FinObj**	**ECMt (s)**	**PBMt (s)**
Hi-tech	33	91	5	88.3%	75.2%	**0.12**	0.2	55.5%	**0.59**	0.64
Karate	34	78	5	85.6%	57.8%	0.16	**0.13**	26.2%	**0.11**	**0.11**
Mexican	35	117	4	98.0%	88.6%	0.31	**0.23**	60.2%	**0.33**	0.39
Sawmill	36	62	8	63.0%	34.1%	0.08	**0.06**	21.4%	**0.08**	**0.08**
Chesapeake	39	170	3	100.0%	93.9%	**0.6**	0.61	69.1%	1.36	**1.18**
Dolphins	62	159	8	58.5%	43.4%	1.91	**1.03**	30.8%	1.91	**1.71**
Lesmiserable	77	254	5	85.4%	31.8%	**0.52**	0.92	11.0%	**0.97**	1.03
Santafe	118	200	12	32.9%	4.4%	**0.2**	0.45	1.7%	**0.59**	0.72
Sanjuansur	75	155	7	48.7%	28.9%	0.44	**0.35**	16.5%	**0.39**	0.58
Attiro	59	128	8	68.0%	43.4%	0.39	**0.31**	26.0%	0.67	**0.61**

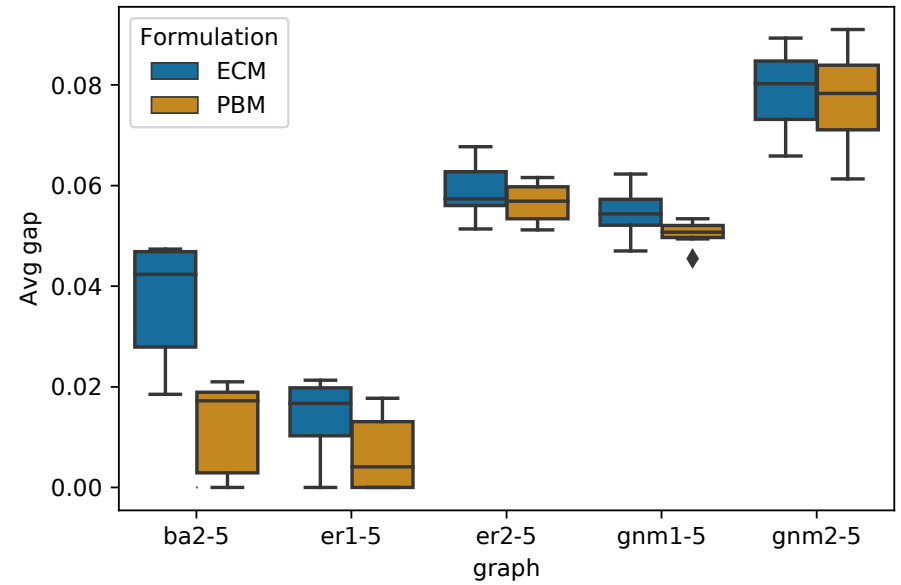
Table 3.3: Results of ECM and PBM models with DCNDP-1 on realworld networks (small size)

Tables 3.3 & 3.4 summarise results for real-world network instances. What we see from Table 3.3 is that the path based model (PBM) competes well with the base model

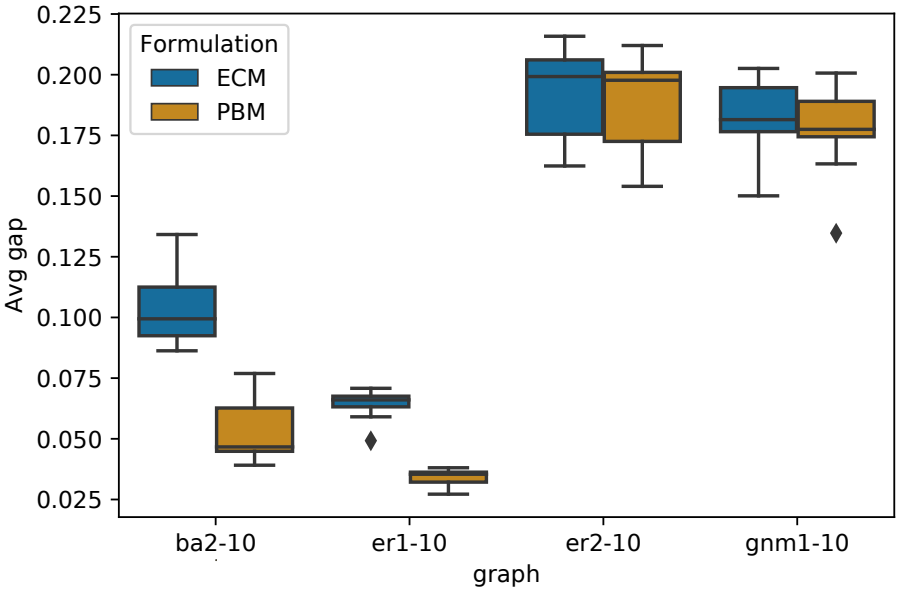
Graph	n	m	diam	InitObj	B	FinObj	ECMt (s)	PBMt (s)
USAir97	332	2126	6	84.8%	0.1n	5.64%	> 3600	**1277.38**
					0.05n	19.33%	426.88	**377.83**
					0.03n	39.35%	952.22	**692.04**
					0.02n	49.57%	**515.49**	582.16
					0.01n	63.90%	456.41	**240.67**
LindenStrasse	232	303	13	12.1%	0.04n	4.70%	**0.91**	1.17
					0.03n	6.15%	**0.88**	1.82
					0.015n	8.32%	**0.86**	1.25
SmallWorld	233	994	4	95.2%	0.1n	6.27%	144.56	**117.54**
					0.04n	19.45%	78.66	**53.61**
					0.03n	23.41%	46.75	**18.46**
					0.015n	40.56%	85.17	**41.17**
NetScience	379	914	17	13.3%	0.03n	4.32%	**4.08**	5.11
					0.01n	8.43%	4.72	**4.52**

Table 3.4: Results of ECM and PBM models with DCNDP-1 on real world networks (medium size)

for small graph sizes (< 120 nodes). As the real-world network instances increase in size with increase in initial objective, the performance of PBM over ECM becomes more glaring. For example, in **Smallworld** and **USAir97** graphs with initial $\%k$-distConn greater than 80%, the PBM on average is almost twice as fast as the ECM. In particular, for budget setting of 0.1n, ECM fails to solve the **USAir97** instance terminating with a 10.3% gap whereas this is solved under 1300 seconds by PBM (see Table 3.4). This is also the case for the smaller class of Barabasi-Albert random network (**ba1)**, in which PBM is more than three times as fast as the ECM for B=0.05n (**ECMt=1263.18s, PBMMt=374.33s**) and more than twice as fast for B=0.1n (**ECMt=1158.20s, PBMt=418.94s**). For the rest of the synthetic networks, both models are unable to solve these instances, thus we compare the average percentage gaps for both models. The instances are labelled **network-budget**. For instance, **er1-5** represents the **er1** network with 0.05n budget setting. From the graphs in Figure 3.2, the average %gap for the ECM is larger than those of PBM. In particular, the %gap for ECM is twice that of PBM for **ba2-5 (ECM=3.75%, PBM=1.24%)**; **ba2-10 (ECM=10.32%, PBM=5.35%)** and **er1-10 (ECM=6.44%, PBM=3.41%)**. Moreover, for **ba2-5** and **er1-5** random network classes, we observed that among the 10 instances generated for each class, PBM successfully solved 30% and 50%, respectively, while the success for ECM was 0% and 20%, respectively. Furthermore, percentage gap is observed to increase with the budget for both models. The difference in average % gap for both models is seen to be even more pronounced for larger instances of the uniform random graph model (see Figure 3.3). This is understandable as these instances are larger than the rest both in terms of number of nodes and edges. Moreover, the topology of the network is characterised by a small diameter and a large proportion of nodes being connected by very few hops. Similar to other random network instances, the average percentage gap increases with the budget for both models. For all instances and budget setting, PBM is seen to outperform ECM. PBM in conjunction with the BFS cuts excelled particularly for large edge densed instance (n=300, m=2000) where almost all node pair connections are within short distances. We see the ECM struggling to solve the LP relaxations as



(a) Budget=$0.05n$



(b) Budget=$0.1n$

Figure 3.2: Variation of average gaps of ECM and PBM models for DCNDP-1 on random network. Average gaps are taken over 10 problem instances for each network class.

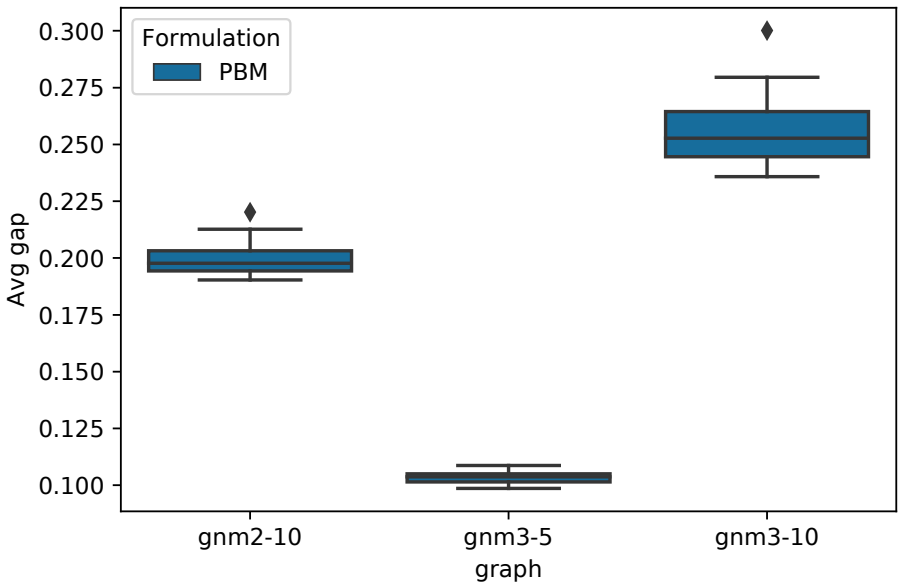


Figure 3.3: Variation of average gaps of PBM for DCNDP-1 on uniform random network. ECM gap remains at 100% for all instances of gnm3-5 and gnm3-10. It solves 50% of gnm2-10 graphs to an average gap of 20% and the rest of instances have 100% gap.

can be seen from the average gap of 100%, whereas with our approach, we are still able to achieve competitive bounds and gaps in these instances.

3.6.2 Results of the base and the path formulations for DCNDP-2

We now present results of computational experiments for the DCNDP-2 (that is, distance connectivity function 3.2). Recall that for this class, the objective is to minimise the

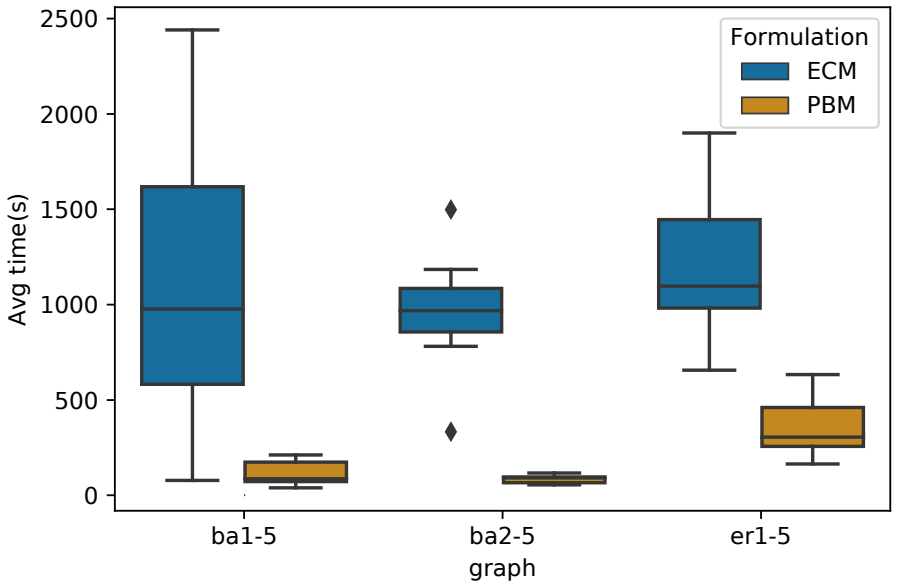


Figure 3.4: Variation of average computational times of ECM and PBM models for DCNDP-2 on Barabasi-Albert and Erdos-Renyi network instances for B=0.05n

communication efficiency. We use the same set of real-world graphs with budget setting of 5% and 10% of graph size. For randomly generated graphs, we use the same instances of Barabasi-Albert and Erdos-Renyi classes but with budget setting of only 5% of graph size due to the observation of computational difficulty when the budget increases. Results for these experiments are summarised in Table 3.5 for real-world network instances and in Figure 3.4 for random network instances. Analysing results for the random network instances, it can be seen from Figure 3.4 that PBM is on average 3 times faster than

the ECM for all three sets of instances. Moreover for the larger Erdos-Renyi class

					B=0.05n			B=0.1n		
Graph	**n**	**m**	**diam**	**InitObj**	**FinObj**	**ECMt (s)**	**PBMt (s)**	**FinObj**	**ECMt (s)**	**PBMt (s)**
Hi-tech	33	91	5	50.8%	43.69%	0.47	**0.45**	32.81%	2.38	**1.53**
Karate	34	78	5	49.2%	33.74%	0.3	**0.19**	16.69%	0.38	**0.28**
Mexican	35	117	4	55.0%	49.06%	**0.35**	0.41	36.58%	0.92	**0.71**
Sawmill	36	62	8	39.9%	27.46%	0.75	**0.44**	14.17%	0.67	**0.34**
Chesapeake	39	170	3	60.4%	53.71%	**0.25**	0.27	35.87%	**0.33**	0.69
Dolphins	62	159	8	37.9%	29.33%	243.13	**23.03**	18.63%	609.81	**24.73**
Lesmiserable	77	254	5	43.5%	18.44%	**3.02**	3.85	7.88%	7.27	**5.02**
Santafe	118	200	12	27.0%	2.95%	14.07	**3.41**	1.39%	21.39	**4.83**
Sanjuansur	75	155	7	34.2%	25.90%	141.1	**25.96**	14.41%	224.22	**28.62**
Attiro	59	128	8	38.9%	31.11%	29.47	**6.17**	22.30%	175.81	**41.93**
SmallWorld	233	994	4	45.4%	9.28%	642.7	**264.88**	4.02%	> 3600	**822.96**
NetScience	379	914	17	20.3%	2.09%	M	**1166**	0.94%	M	**181.55**

Table 3.5: Results of ECM and PBM models with DCNDP-2 on realworld networks

(er2-5), a striking observation is that while PBM achieves an average percentage gap of 6.56%, ECM is seen to struggle with solving the linear programming relaxation of these instances within the given time limits. We notice a similar trend for the real-world graphs, PBM competes well with ECM for the smaller instances and indeed performs a lot better in terms of computational times as the graph size increases (see Table 3.5). For instance, for **Dolphins** network, PBM is more than 10 times faster for both budget settings $B = 0.05n$ and $B = 0.1n$. Similarly, for **Santafe, Sanjuansur & Attiro** networks, PBM is atleast 4 times faster than ECM for both budget settings. Moreover, for the **NetScience graph** which has a large diameter (diameter=17), we see that while PBM is able to solve the problem to optimality in 1166 and 182 seconds respectively for budget settings $B = 0.05n$ and $B = 0.1n$, the ECM runs out of memory. This behaviour might be related to the issue of sensitivity to large diameter of the base model (ECM) as reported in Hooshmand et al. (2020) for a different class of the DCNDP (Wiener index). For **USAir97** and **LindenStrasse**, both models fail to close the gap within the time limit. In particular, with **LindenStrasse**, both ECM and PBM struggle to obtain good

dual bounds leading to gaps of 108.6% and 102.3%, respectively. Moreover, comparison with the results obtained for the same graphs earlier reported for the first distance class provides strong insights for our discussion in Section 3.6.4.

3.6.3 Results of path-based model with inclusion of oddhole inequalities and primal heuristics

We now examine the impact of the proposed oddhole inequalities and primal heuristic presented in Sections 3.4.2 & 3.4.3. We compare the performance of the path-based models with and without the inclusion of the oddhole inequalities and primal heuristics. The former is labelled as *PBM+oddhole+heur* while the latter remains as *PBM*. We focused our computational tests on those instances which were unsolved by PBM within the time limit This consists of the random graph instances with $B = 0.1n$ for DCNDP-1 and **USAir97** and **LindenStrasse** for DCNDP-2 with $B = 0.05n$. Results are reported in Table 3.6 for network classes where there is a significant difference between *PBM* and *PBM+oddhole+heur*. We omit results for uniform random graphs (**gnm1, gnm2 & gnm3** instances) and Erdos-Renyi graph **er2** where no improvement was observed. From the table, we observe some improvement of version *PBM+oddhole+heur* with respect to *PBM* over the denser Barabasi and Erdos-Renyi graph classes (**ba2** & **er1**) instances, where tighter bounds are realised with the inclusion of the odd hole inequalities. However, across all of the uniform random graphs (**gnm1, gnm2 & gnm3**) instances as well as over Erdos-Renyi **er2** instances, *PBM* and *PBM+oddhole+heur* behave in a similar way. A possible explanation for this is the very few number of cycles reported in these graphs which could have reduced the chances of having violated inequalities. With respect to the cycle enumeration, we observed that about four times as many cycles were enumerated for the Barabasi graph class **ba2** and twice as many cycles for Erdos-Renyi **er1** thus leading to more usercuts violations. A clear evidence of this structural difference is in the edge densities of these graph classes. While **er1** and **ba2** instances have average densities of 14.9% and 18.2% respectively, the **gnm1, gnm2, gnm3** and **er2** instances have smaller average densities of 5.0%, 3.3%, 4.5% and 5% respectively. For the

				PBM			PBM+oddhole+heur		
Instances	**n**	**m**	**density**	**LB**	**UB**	**%gap**	**LB**	**UB**	**%gap**
er1(1)	80	456	14.4%	2400.65	2466	2.72%	**2402.94**	2466	**2.62%**
er1(2)	80	472	14.9%	2398.17	2484	3.45%	**2408.61**	**2481**	**3.01%**
er1(3)	80	474	15.0%	2394.56	**2474**	3.32%	**2404.32**	2483	**2.90%**
er1(4)	80	446	14.1%	2383.29	2474	3.81%	**2393.60**	2474	**3.36%**
er1(5)	80	475	15.0%	2398.21	2483	3.45%	**2413.72**	**2481**	**2.79%**
er1(6)	80	476	15.1%	2394.74	2482	3.60%	**2413.25**	**2481**	**2.81%**
er1(7)	80	474	15.0%	2395.02	2484	3.59%	**2410.33**	**2481**	**2.93%**
er1(8)	80	471	14.9%	2393.31	**2479**	3.58%	**2407.59**	2481	**2.97%**
er1(9)	80	447	14.1%	2378.37	2452	3.10%	**2403.22**	2452	**2.03%**
er1(10)	80	505	16.0%	2398.67	2475	3.14%	**2417.92**	**2474**	**2.32%**
ba2(1)	100	900	18.2%	3674.25	4005	6.58%	**3735**	**4004**	**4.85%**
ba2(2)	100	900	18.2%	3744	**3914**	4.54%	**3753.14**	3916	**4.29%**
ba2(3)	100	900	18.2%	3716	4004	7.29%	**3722.8**	**4003**	**7.10%**
ba2(4)	100	900	18.2%	3751.18	3913	4.31%	**3759.25**	3913	**4.09%**
ba2(5)	100	900	18.2%	3737	**3916**	4.79%	**3741.44**	4003	**4.67%**
ba2(6)	100	900	18.2%	3717.45	3916	5.26%	**3745.66**	**3913**	**4.47%**
ba2(7)	100	900	18.2%	3745.14	3915	4.51%	**3769**	**3914**	**3.85%**
ba2(8)	100	900	18.2%	3767.83	3915	3.85%	**3790.42**	**3913**	**3.23%**
ba2(9)	100	900	18.2%	3702.2	4005	5.77%	**3755.38**	**3916**	**4.28%**
ba2(10)	100	900	18.2%	3744.49	**3912**	4.47%	**3744.76**	3914	**4.47%**
USAir97	332	2126	3.9%	5703.83	10745.58	88.39%	**5758.62**	10745.58	**86.60%**
LindenStrasse	232	303	1.1%	**1162.01**	2350.18	**102.25%**	1150.86	2350.18	104.21%

Table 3.6: Results of PBM and PBM+oddhole+heur

DCNDP-2 instances, *PBM+oddhole+heur* improved in lower bounds for the **USAir97** graph instance but fell behind its counterpart *PBM* for **LindenStrasse** graph.
With regards to the primal bounds, better upper bounds were recorded for some of the **ba2** and **er1** instances. Although this improvement is mild, it is indicative of the potential of the primal heuristics.

3.6.4 Result of the path based model with different connectivity metrics

One of the insights drawn from an earlier research on the CNDP is that the ease of solving an instance of the CDNP is influenced by the connectivity metric, the specified budget as well as the topology of the input network (Veremyev, Prokopyev & Pasiliao 2014). We investigate this by comparing results for different connectivity metrics on a range of problem instances varying in size of networks and edge density. We begin by investigating the behaviour of both the base model and the path based models on the first two DCNDP classes based on results for the real-world instances (see Tables 3.3, 3.4 & 3.5). The first observation is that both the path-based model and the base model are more difficult to solve for DCNDP-2 i.e distance connectivity metric (3.2) than for DCNDP-1 distance connectivity metric (3.1). For example, for the **SmallWorld** graph with 10% budget, Gurobi is able to solve DCNDP-1 in less than 150 seconds (Table 3.4) whereas it takes over 3600 and 800 seconds (Table 3.5) to solve the base model and path-based model respectively for DCNDP-2. Similarly, for the **USAir97** graph with 5% budget, Gurobi is unable to solve both the base model and path-based model for DCNDP-2. However with DCNDP-1, both models are solved under 430 seconds. Even more striking difference is observed for the **Lindenstrasse** and **NetScience** graphs, even though both graphs have very small initial communication efficiency (20%) and k-DistConn (13%). For the random networks, however, DCNDP-2 appears to be less computationally demanding than DCNDP-1. For example, while both the compact and path-based models were solved to optimality on just a single random network type **(ba1)** for DCNDP-1, DCNDP-2 was solved for two additional random network types **(ba2-5 & er1-5)**. These differences in computational ease confirm earlier observations that the

choice of objective metric is an important part of the critical node detection problem.

We conclude this section by extending this investigation to two additional connectivity metrics. The first is class 3 of the DCNDP whose distance function (3.3) minimises the sum of power functions of distances. The other is the classical fragmentation-based CNDP which minimises the total number of connected node pairs. We denote the problems corresponding to these metrics by DCNDP-3 and CNP respectively. For all CNDP variants, the path-based formulation with the corresponding connectivity objective metric is used. Results are summarised in Table 3.7. The instances are labelled as *Graph-budget* for example, *Dolphin-5* instance is the Dolphins graph with budget on the critical node set specified as 5% of the graph size. Columns labelled *Initial* and *Final* denote the initial and final objective value in percentage before and after node deletion. The running times are specified in the columns labelled *time*. The best and worst running times for each instance are highlighted in blue and red respectively. When the optimisation process does not terminate within the specified time limit of 3600seconds, we specify this with "limit" in the respective entry.

From Table 3.7, we observe that the behaviour in terms of computational time across all four connectivity metrics is similar for the smaller real-world instances but as the graph size increases, we observe notable differences. For example, while all 3 DCNDP classes were solved for the **Smallworld-10** instance, the instance is unsolved with the fragmentation-based CNP. This is not strange since the **Smallworld-10** is a larger instance of a small world citation network characterised by small hop distances with diameter equal to 4. Hence, with L fixed to the diameter (that is, $L = 4$) and k fixed to 3, the instance is easily solve by all distance-based CNDP. For the **USAir97** instances, a more interesting observation is made. We see that while the smaller **USAir97** instance (*USAir97-5*) is solved under 400 seconds using DCNDP-1, it is 7 times more computationally demanding to solve with any of the other metrics. The computational competitiveness is even more pronounced for the larger instance (*USAir97-10*), which remains unsolved using any of the other 3 metrics whereas with DCNDP-1, it is solved under 1300 seconds. Considering that these instances come from a transportation

				DCNDP-1			DCNDP-2			DCNDP-3			CNP		
Instance	**n**	**m**	**diam**	**Initial**	**Final**	**time**	**Initial**	**Final**	**time**	**Initial**	**Final**	**time**	**Initial**	**Final**	**time**
Mexican-5	35	117	4	98.0%	88.6%	0.2	55.0%	49.1%	0.4	26.3%	22.9%	0.4	100%	94.3%	1.5
Mexican-10	35	117	4	98.0%	60.2%	0.4	55.0%	36.6%	0.7	26.3%	16.4%	0.8	100%	73.3%	1.8
Chesapeake-5	39	170	3	100.0%	93.9%	0.6	60.4%	53.7%	0.3	29.9%	26.2%	0.3	100%	94.9%	1.5
Chesapeake-10	39	170	3	100.0%	69.1%	1.2	60.4%	35.9%	0.7	29.9%	16.7%	0.6	100%	80.3%	1.7
Dolphins-5	62	159	8	58.5%	43.4%	1.0	37.9%	29.3%	23.0	15.1%	11.6%	14.1	100%	75.6%	19.3
Dolphins-10	62	159	8	58.5%	30.8%	1.7	37.9%	18.6%	24.7	15.1%	8.5%	45.6	100%	37.3%	4.9
LesMiserable-5	77	254	5	85.4%	31.8%	0.9	43.5%	18.4%	3.9	19.1%	8.3%	3.8	100%	37.6%	2.3
LesMiserable-10	77	254	5	85.4%	11.0%	1.0	43.5%	7.9%	5.0	19.1%	3.7%	3.4	100%	13.2%	2.8
Smallworld-5	233	994	4	95.2%	17.1%	202.9	45.4%	9.3%	264.9	21.1%	3.9%	193.9	100%	22.7%	443.2
Smallworld-10	233	994	4	95.2%	6.3%	117.5	45.4%	4.0%	823.0	21.1%	1.6%	203.8	100%	13.7%	limit
USAir97-5	332	2126	6	84.8%	19.3%	377.8	40.6%	19.6%	limit	17.5%	4.7%	2975.0	100%	28.1%	2795.1
USAir97-10	332	2126	6	84.8%	5.6%	1277.4	40.6%	3.8%	limit	17.5%	1.8%	limit	100%	19.4%	limit
ba1(1)-5	100	475	4	99.9%	87.2%	571.5	49.5%	40.6%	148.1	23.4%	18.3%	53.7	100%	88.3%	1981.9
ba1(1)-10	100	475	4	99.9%	68.2%	395.6	49.5%	33.0%	1535.3	23.4%	14.1%	190.6	100%	77.3%	limit
er1(1)-5	80	470	3	100.0%	87.8%	296.6	54.5%	48.0%	317	26.6%	23.2%	74.8	100%	87.8%	226.6
er1(1)-10	80	470	3	100.0%	78.0%	limit	54.5%	41.5%	1783.9	26.6%	20.0%	921.9	100%	78.6%	limit

Table 3.7: Comparison of various connectivity metrics: **DCNDP-1** that is $f(d) = 1$ for $d \leq k$ and $f(d) = 0$ for $d > k$; **DCNDP-2** that is, $f(d) = \frac{1}{d}$ for $d \leq L$ and $f(d) = 0$ for $d > L$; **DCNDP-3** that is, $f(d) = \frac{1}{2}^d$ for $d \leq L$ and $f(d) = 0$ for $d > L$: **CNP** that is, $f(d) = 1$ for $d \leq n-1$ and $f(d) = 0$ for $d \geq n$. We set $k = 3$ and L =diameter(G).

network of airlines where most flights are direct, 1 or 2 layovers, the DCNDP-1 would be a more appropriate connectivity objective. The interpretation of the problem being intuitive in the sense that the critical nodes are those airlines which if deleted (that is, not in operation for whatever reason) would reduce the number of direct, 1 or 2 layover flights to a minimum.

Overall, we see from the results in Table 3.7, that the fragmentation-based CNP metric proves to be computationally more demanding for graphs with smaller diameter (3 or 4) but quite competitive with its distance-based counterparts for the instances with larger diameter. Conversely, DCNDP-2 and DCNDP-3 metrics seem to be computationally favorable for graphs with small diameter, however, they both begin to struggle as the diameter increases. Moreover, the DCNDP-3 metric appear to be the most computationally favorable metric for both the Barabasi-Albert and Erdos-Renyi graph instances while the CNP performed worst across these random graph types. On average, DCNDP-1 metric is more competitive both for instances with small and large diameter.

3.7 Concluding remarks

In this chapter, we considered the hop-distance version of the distance-based critical node detection problem. We presented new mixed integer programming formulations along with efficient separation heuristics and strong valid inequalities that exploit the structure of the problem. Extensive computational experiments on both real-world and synthetic graphs based on the first two classes of the distance-based critical node detection problem shed light on the scalability of our approach. The effectiveness of our approach is more evident as the graph size grows, when the existing compact model struggles to solve even the linear programming relaxation. The computational experiments on four different connectivity metrics also provide insights into the influence of graph topology and objective metric on the identification of critical nodes in a network. This emphasises the need for appropriate choice of objective metric for specific application setting in order to design efficient solution methods. In the next chapter, we propose a heuristic

framework to handle larger instances of the problem for which the exact approaches struggle.

Chapter 4

Heuristic algorithm for the distance-based critical node detection problem

4.1 Introduction

Traditional exact algorithms such as branch-and-bound, branch-and-cut and Benders decomposition have been employed to solve the critical node detection problem (see for example, Arulselvan et al. (2009), Di Summa et al. (2012), Veremyev, Boginski & Pasiliao (2014), Hooshmand et al. (2020), Alozie et al. (2021)). However, due to its combinatorial nature, the complexity of the critical node detection problem grows significantly with the size of the network. This limits the success of exact solution methods to small and medium size instances of the problem. To mitigate this gap, heuristic algorithms have been developed to provide good solution to larger instances of the problem. For example, Ventresca & Aleman (2015) proposed a greedy heuristic algorithm that is based on a modified depth first search. Two new neighbourhoods were developed by Aringhieri et al. (2016*b*) and used within a variable neighbourhood search solution framework. The new neighbourhoods are more computationally efficient than the traditional two node

exchange. Aringhieri et al. (2016*a*) proposed a genetic algorithm for the classical CNDP as well as its cardinality-constrained variant. A Greedy Randomised Adaptive Search Procedure (GRASP) with Path Relinking (PR) mechanism was proposed for the classical CNDP by Purevsuren et al. (2016). Recently, Zhou et al. (2018) developed a memetic algorithm for both the classical and the cardinality-constrained variants of the CNDP. For a detailed discussion on heuristic solution methods as well as current developments in their application to combinatorial optimisation problems, we refer the reader to Silver (2004) as well as Aickelin & Clark (2011). To the best of our knowledge, the memetic algorithm proposed by Zhou et al. (2018) is the current state-of-the-art heuristic algorithm for the classical critical node detection problem based on computational experiments on 26 real-world and 16 synthetic benchmark instances. All the aforementioned studies on heuristic methods for the critical node detection problem focus on the fragmentation-based variants of the critical node detection problem. To this end, this chapter addresses the distance-based critical node detection from a heuristic perspective. We propose a heuristic framework for the first class of the DCNDP whose goal is to minimise the number of node pairs connected by a distance of length at most k. We demonstrate the efficiency of our proposed algorithm in comparison to the exact approaches discussed in Chapter 3 on both real-world and synthetic graphs. We also show how our proposed heuristic framework can be extended to other classes of the DCNDP.

Contributions

Our main contributions consist of the following:

i. We describe a new heuristic algorithm for the distance-based critical node detection problem. The feasible solution construction procedure utilizes centrality measures along with the idea of backbone-based crossovers to construct good feasible solutions. The neighbourhood search procedure uses a newly developed two-stage node exchange strategy to focus local search on a reduced centrality-based neighborhood thus making the search more efficient.

ii. The proposed algorithm yields competitive results on both real-world and synthetic graphs. In particular for the real-world instances, our heuristic algorithm matches the exact optimal solutions for all 28 instances. For the synthetic graphs which comprise of 54 instances, the heuristic achieves the optimal objective value or best known upper bounds on 10 of the instances and discovers new upper bounds on 33 of the instances within very short time duration in comparison to the exact methods.

iii. Our empirical results provide useful insights regarding the effect of topological structures of certain model networks on algorithm behaviour.

Organisation

The rest of the chapter is structured as follows. In Section 4.2, we recapitulate the description of the distance-based critical node detection problem with definition of the distance connectivity function of interest. Section 4.3 describes the proposed heuristic algorithm in detail as well as ideas on how to extend the proposed framework to other classes of the DCNDP. In Section 4.5, parameter settings and results of our computational experiments are presented. For real-world and synthetic network instances, we compare the performance of our proposed heuristic algorithm with results of the exact methods. We end the chapter with some concluding remarks in Section 4.6.

4.2 Problem Description

Given an input graph $G = (V, E)$ with $n = |V|$ nodes (vertices) and $m = |E|$ edges, as well as a positive integer B, the distance-based critical node detection problem aims to find a subset of nodes S of cardinality at most B, whose removal minimises a certain distance-based connectivity objective. Five different distance-based connectivity functions were defined by Veremyev et al. (2015) all of which have been described in Chapter 3. In this chapter, we focus on the first distance-based connectivity objective which minimises the number of node pairs connected by a path of length at most k.

This class of the DCNDP denoted as DCNDP-1 has interesting real life applications, for example in transportation engineering and is the most studied. Recall the definition of the distance function for DCNDP-1:

$$f(d) = \begin{cases} 1, & \text{if } d \leq k \\ 0, & \text{if } d > k \end{cases} \tag{4.1}$$

where d is the distance (shortest path length) between node pairs in the induced subgraph $G^S = G[V \setminus S]$, and k is a given positive integer representing the cut-off distance.

4.3 Heuristic for distance-based critical node detection problem

This section describes the proposed heuristic algorithm for the distance-based critical node detection problem. The underlying idea of the proposed heuristic framework is akin to the memetic algorithm proposed for the classical critical node problem by Zhou et al. (2018) however with some striking differences as described later in the chapter (Section 4.3.4). Memetic algorithm has seen successful applications in solving $\mathcal{NP}$-hard problems (Pereira et al. 2018, Du et al. 2017, Yadegari et al. 2019, Wang et al. 2020). For a review of memetic algorithm and its application to several classes of optimisation problem, we refer the reader to the survey by Neri & Cotta (2012).

4.3.1 Representation and evaluation of feasible solution

Given an input graph $G = (V, E)$, a feasible solution to the DCNDP is any collection of B distinct nodes. For any feasible solution S, the objective function value $f(S)$ according to equation (4.1) evaluates the number of node pairs connected by hop distance less or equal to k in the induced subgraph $G^S = G[V \setminus S]$. By running an all-pairs shortest path algorithm on $G[V \setminus S]$, one can calculate $f(S)$ by counting the number of such pairs whose shortest path length is less than or equal to k. The fastest of such algorithms requires $O(|V|^3)$ time which is quite expensive given that the objective function would be evaluated multiple times for different feasible solution. Instead, we compute $f(S)$ by generating a k-depth breadth-first-search tree for each node. The $k-$depth BFS tree

runs the general breadth-first-search up to a given depth k (see Figure 3.1). The size of each tree gives the number of nodes connected to the root node within a hop distance at most k. The time complexity of general BFS tree generation is $O(|V||E|)$. Note that this complexity can be significantly improved to $O(b^k)$, when we are restricting BFS trees to depths of k, where b is the branching factor (or average outdegree) of the tree. For small values of k, this reduces to linear time complexity and it empirically makes an immense difference.

4.3.2 General framework of heuristic

The proposed heuristic algorithm consists of three components: an initial solution generation procedure, a backbone-based crossover and a centrality-based neighbourhood search procedure.

Algorithm 2: The proposed heuristic algorithm for DCNP

Input : Graph $G = (V, E)$, an integer B
Output: the best solution S^* found
`/* construct initial centrality-based solutions, section 4.3.3 */`
$P^0 = \{C_1, C_2, C_3\} \leftarrow$ centralitysolution();
$S^* = \text{argmin}\{f(C_1), f(C_2), f(C_3)\}$
`/* generate offspring solution, section 4.3.4 */`
$S^1 \leftarrow$ backboneCrossover(C_1, C_2, C_3)
`/* perform local search, section 4.3.5 */`
$S^+ \leftarrow$ neighbourhoodSearch(S^1);
if $f(S^+) < f(S^*)$ **then**
    $S^* = S^+$
else
    pass;
end

The algorithm begins with the generation of centrality-based solutions. An improved

offspring solution is then generated from these centrality-based solutions through a backbone-based crossover (Section 4.3.4). This offspring solution is further improved by a centrality-based neighbourhood search procedure (Section 4.3.5). A pseudocode of the general framework of the proposed algorithm is presented in Algorithm 2. Its three components are described in detail in the subsequent sections.

4.3.3 Initial solution generation

The proposed algorithm begins with construction of initial feasible solutions using three centrality metrics. The first is the popular degree centrality where nodes are ranked according to their degrees. The other two measures could be seen as specialised adaptations of the Katz and betweenness centralities. The first which we refer to as $k-$Katz centrality ranks nodes according to the size of the k-depth breath first search (BFS) tree rooted at each node. The last centrality metric which we refer to as k-betweenness ranks nodes according to the number of their direct offspring summed over all generated k-depth BFS trees. We refer the interested reader to Paton et al. (2017) for detailed discussion and a numerical analysis of centrality measures.
Let v be an arbitrary node in a graph, we summarise the centrality definitions as follows:

Definition 4.3.1. *(**degree centrality**): The degree of a node v is the number of edges incident on v, i.e., the number of direct neighbours of v.*

Definition 4.3.2. *($k-$**Katz centrality**): The $k-$Katz of v is the number of nodes reachable from v at a hop distance less than or equal to k.*

Definition 4.3.3. *($k-$**betweenness centrality**): The $k-$betweenness of v is the number of direct offsprings of v summed across all generated k-depth BFS trees.*

Let $\mathcal{R}_1, \mathcal{R}_2$ and $\mathcal{R}_3$ denote 3 different collections of all nodes in the input graphs ordered according to the three defined centrality measures. From each of these collections, we generate 3 feasible centrality-based solutions $\mathcal{C}_1, \mathcal{C}_2$ and $\mathcal{C}_3$ through a probabilistic selection of B nodes. For example, to generate $\mathcal{C}_1$, we sequentially add each node in $\mathcal{R}_1$ into $\mathcal{C}_1$ with probability $p = 0.90$ until the required budget is attained. We also extend

the budget value by a certain number, $\epsilon = max(5, 0.02B)$ and then select the next ϵ top nodes in each of $\mathcal{R}_1, \mathcal{R}_2$ and $\mathcal{R}_3$. The union of these extended budget solutions which we denote by X_ϵ is used in the backbone crossover phase (details in Section 4.3.4).

4.3.4 Backbone-based crossover

Our notion of backbone crossover is similar to the double backbone crossover introduced by Zhou et al. (2018) in the sense that the offspring solution inherits elements that are common to its parent solutions as well as exclusive elements. However, our backbone procedure differs from that of Zhou et al. (2018) primarily in the number of parent solutions used. Secondly, our procedure for repairing a partial offspring solution combines both greedy and random node selection. This combination of greedy and random selection can be seen as a double-edged sword that intensifies and diversifies the node selection. The motivation for the use of three rather than two parent solutions is to limit the members of the partial solution inherited from the parent solutions to only promising nodes. This is potentially useful in arriving at high quality offspring solutions leading to fewer iterations of local search to converge to local optimum. For the backbone crossover, we divide the elements of the centrality-based solutions into sets as follows:

Definition 4.3.4. *(**3-parent elements**): These consist of the intersection of all three centrality-based feasible solution sets, denoted by* $X_1 = \mathcal{C}_1 \cap \mathcal{C}_2 \cap \mathcal{C}_3$.

Definition 4.3.5. *(**2-parent elements**): These consist of elements that are only present in exactly two parent solutions denoted by* $X_2 = ((\mathcal{C}_1 \cap \mathcal{C}_2) \cup (\mathcal{C}_1 \cap \mathcal{C}_3) \cup (\mathcal{C}_2 \cap \mathcal{C}_3)) \setminus X_1$.

Definition 4.3.6. *(**1-parent elements**): These consist of elements that are only present in exactly one parent solution denoted by* $X_3 = (\mathcal{C}_1 \cup \mathcal{C}_2 \cup \mathcal{C}_3) \setminus (X_2 \cup X_1)$.

Definition 4.3.7. *(**0-parent elements**): These consist of elements in the extended budget solutions denoted by* X_ϵ.

The backbone crossover procedure proceeds as follows: An offspring solution S^1 is constructed by first inheriting all elements common to its parent solutions that is,

$S^1 \leftarrow X_1$. If $|S^1| < B$, we repair S^1 by sequentially adding elements from sets X_2, X_3, X_ϵ until the budget is satisfied. At each iteration of the repair process, a new node is selected into the offspring solution by one of greedy or random approaches according to some specified probabilities P_{greedy} and P_{random}. The greedy approach entails selecting a node $u \notin S^1$ which gives the best improvement to the current objective function value $f(S^1)$, i.e $u = \text{argmax}\{f(S^1) - f(S^1 \cup \{v\})\}, \forall v \in (X_2 \cup X_3 \cup X_\epsilon) \setminus S^1$. The random approach uses specified probabilities p_2, p_1 or p_0 to determine which of the sets X_2, X_3 or X_ϵ from which a node is to be chosen at random.

4.3.5 Centrality-based neighbourhood search

We explore the neighbourhoods of the solution realised from the construction phase with the aim of arriving at the optimal solution in the region. We discuss next the neighbourhood structure as well as the node swap technique defined for our study.

Centrality-based neighbourhood structure

Considering the time complexity for evaluation of a candidate solution, the traditional neighbourhood which swaps each node $v \in S$ with a node $u \in V \setminus S$ requires a time of $O(B(|V|-B)(|V||E|))$ to evaluate all neighbourhood solutions. This becomes prohibitive when $|V|$ is very large or when many local search iterations are required. To mitigate this time complexity, we design a much smaller alternative neighbourhood which also exploits the structure of the objective function. Let s be a positive integer corresponding to the size of each centrality-based neighbourhood. Thus our centrality based neighbourhood N_s for a given solution S consists of the union of the top s nodes ranked according to the three defined centrality measures in the residual graph $G[V \setminus S]$. Similar to the generation of centrality-based solutions, the top ranking nodes in each centrality measure have a 90% chance of being selected into the corresponding centrality neighbourhood. The cardinality of N_s is bounded below and above by s and $3s$. Hence, the size of the neighbourhood is reduced to $(B|N_s|)$.

Algorithm 3: Centrality-based neighbourhood search

Input : a starting solution S, centrality-based neighbourhood N_s, size of centrality neighbourhood s

Output: the best solution S^* found

$S^* \leftarrow S$;

$iterCnt \leftarrow 0$;

while $iterCnt < maxIter$ *and* $N_s \neq \emptyset$ **do**

  $v \leftarrow N_s.remove()$;

  $S \leftarrow S \cup \{v\}$;

  $u \leftarrow \text{argmin}_{w \in S} \{f(S \setminus \{w\}) - f(S)\}$;

  $S \leftarrow S \setminus \{u\}$;

  if $f(S) < f(S^*)$ **then**

    $S^* = S$;

    $iterCnt \leftarrow 0$

  end

  else

    $iterCnt \leftarrow iterCnt + 1$;

  end

end

Two-phase node swap

We employ the two-phase node exchange strategy used in Zhou et al. (2018). In keeping with its name, the *two-phase* node exchange strategy is composed of two separate phases: a "removal phase" which removes a node from the resultant subgraph and an "add phase" which adds a node back to the subgraph. At each iteration of the two-phase node exchange strategy, a node is removed from the neighbourhood and added into the current solution S. This makes S infeasible. The second phase repairs this infeasibility by identifying a node $v \in S$ which results in the minimum increase in the objective function value, v is then added back to the subgraph.

4.4 Extension to other classes of the DCNDP

The proposed heuristic framework can easily be extended to other classes of the distance-based critical node detection problem in particular DCNDP class 2 and class 3. Recall that DCNDP-2 and DCNDP-3 respectively minimise the efficiency and residual closeness. Their distance connectivity functions are given respectively by:

$$f(d) = \begin{cases} d^{-1}, & \text{if } d \leq L \\ 0, & \text{if } d > L \end{cases} \qquad \text{and} \qquad f(d) = \begin{cases} p^d, & \text{if } d \leq L \\ 0, & \text{if } d > L \end{cases}$$

The main modification of the heuristics for the above DCNDP distance functions lies in the implementation of the modified BFS tree to evaluate a given feasible solution. Recall that we evaluate a given solution S by generating k-depth BFS trees rooted at each node of the residual graph $G^S = G[V \setminus S]$. For DCNDP-1, it suffices to count the number of nodes on each tree to obtain the corresponding objective value. We can modify the implementation of the tree generation by keeping track of the depth say $l[t]$ of each encountered node t from the root node. Then the objective value can be calculated by summing the values of the distance function $f\left(l[t]\right)$ over every node encountered from the root of each L-depth BFS tree.

4.5 Computational Studies

4.5.1 Test Instances

Our computational experiments were based on both real-world and synthetic network instances. Real-world instances are a subset of networks from the Pajek and UCINET datasets (Batagelj & Mrvar 2006, *UCINET software datasets* n.d.). The first set of synthetic instances consist of Barabasi-Albert, Erdos-Renyi and uniform random graphs which were generated using NetworkX random graph generators (Hagberg et al. 2008). Characteristics of the real-world and NetworkX-generated instances are summarised in Tables (4.1) & (4.2). Detailed descriptions of these network instances are provided in Section 3.5 of Chapter 3.

Graph	**n**	**m**	**diam**	**% k-Conn**
Hi-tech	33	91	5	88.3
Karate	34	78	5	85.6
Mexican	35	117	4	98.0
Sawmill	36	62	8	63.0
Chesapeake	39	170	3	100.0
Dolphins	62	159	8	58.5
Lesmiserable	77	254	5	85.4
Santafe	118	200	12	32.9
Sanjuansur	75	155	7	48.7
Attiro	59	128	8	68.0
LindenStrasse	232	303	13	12.1
SmallWorld	233	994	4	95.2
NetScience	379	914	17	13.3
USAir97	332	2126	6	84.8

Table 4.1: Characteristics of real-world graph instances.

Graph	n	m	diam	density (%)	% k-Conn
ba1	100	475	4.0	9.6	99.9
ba2	100	900	3.0	18.0	100
er1	80	470	3.0	14.9	100
er2	200	1004	4.0	5.0	97.7
gnm1	200	1000	4.2	5.0	97.9
gnm2	300	1500	4.4	3.3	94.1
gnm3	300	2000	4.0	4.5	99.6

Table 4.2: Characteristics of NetworkX-generated synthetic graph instances.

The other set of synthetic networks include instances from the benchmark networks in Ventresca (2012). Since, these benchmark instances were generated for the traditional fragmentation CNDP and not the distance-based variant, only 2 (13%) of these instances have % k-Conn greater than 20%. We use the 2 instances (labelled as **FF250** and **WS250a** in Table 4.6) and generate additional instances of similar size and order as the original benchmark networks in Ventresca (2012).

4.5.2 Experimental settings

Our computational study was performed on an HP computer equipped with Windows 8.1 x64 operating system, an Intel Core i3-4030 processor(CPU 1.90 GHz) and RAM 8GB. The proposed algorithms were implemented in Python 3.6. We used NetworkX (Hagberg et al. 2008) for random graph generation. In the design of the proposed algorithm, some parameters were selected for our computational experiments. We executed preliminary experiments to select most of these parameters. Final values of the parameters used in the computational study presented in this chapter are summarised in Table 4.3. All experiments were run with a time limit of 3600 seconds. In line with previous studies, we also set hop distance threshold $k = 3$ which is reasonable since most of the tested instances have a large proportion of nodes connected within this hop distance.

Parameter	Description	Values
ϵ	extended budget limit for centrality solution	max(5, 0.2B)
s	size of each centrality-based neighborhood	$B + \epsilon$
maxIter	maximum no of improvement iteration	100
P_{greedy}	probability of greedy node selection in any crossover iteration	0.7
P_{random}	probability of random node selection in any crossover iteration	0.3
p_2	probability of random node selection from set X_2	0.5
p_1	probability of random node selection from set X_3	0.3
p_0	probability of random node selection from set X_ϵ	0.2

Table 4.3: Parameter settings for computational experiments.

4.5.3 Performance of the heuristic algorithm

We present results obtained from the proposed heuristic algorithm for different graph instances. The results have been summarised from ten independent runs for each instance. Values reported in the tables include objective function values minimum (*min*), mean (*avg*), maximum (*max*) and standard deviation (*std*) for the proposed heuristic algorithm, as well as optimal objective function values (*Opt*) or best lower bounds (*LB*) and best upper bounds (*UB*) realised from Gurobi solver 8.1.0 (Gurobi Optimization 2018). The reported optimal objective function values or best lower and upper bounds were generated using the integer programming models in Veremyev et al. (2015) and Alozie et al. (2021). Computational times (in seconds) for both exact and heuristic algorithms are also reported in columns labelled respectively by *t_exact* and *t_heur* with the smaller run time highlighted in bold. Exact optimal (*Opt*) or best upper bounds (*UB*) are compared with heuristic minimum objective values (*min*).

Real-world instances

Results for the real-world instances are summarised in Table 4.4, where we observe that the proposed heuristic attains the optimal objective values for all 28 instances. Also the average and maximum objective function values obtained by the heuristic matches the optimal solution for 16 of the instances (see when std=0.0 in Table 4.4).

	B=0.05n							B=0.1n						
Graph	**Opt**	**t_exact**	**min**	**avg**	**max**	**std**	**t_heur**	**Opt**	**t_exact**	**min**	**avg**	**max**	**std**	**t_heur**
Hi-tech	**397**	**0.12**	**397**	397.0	397	0.0	0.2	**293**	0.59	**293**	294.8	297	1.9	**0.5**
Karate	**324**	**0.13**	**324**	324.0	324	0.0	0.2	**147**	**0.11**	**147**	150.9	186	11.7	0.4
Mexican	**527**	**0.23**	**527**	527.0	527	0.0	0.3	**358**	**0.33**	**358**	358.0	358	0.0	0.6
Sawmill	**215**	**0.06**	**215**	215.0	215	0.0	0.2	**135**	**0.08**	**135**	135.0	135	0.0	0.2
Chesapeake	**696**	0.6	**696**	696.0	696	0.0	**0.3**	**512**	1.18	**512**	515.2	528	6.4	**1.1**
Dolphins	**820**	**1.03**	**820**	820.0	820	0.0	1.3	**583**	**1.71**	**583**	591.7	616	13.4	2.5
Lesmiserable	**930**	**0.52**	**930**	930.0	930	0.0	1.9	**323**	**0.97**	**323**	323.0	323	0.0	2.9
Santafe	**305**	**0.2**	**305**	305.0	305	0.0	1.2	**116**	**0.59**	**116**	116.0	116	0.0	2.8
Sanjuansur	**803**	**0.35**	**803**	803.0	803	0.0	0.9	**457**	**0.39**	**457**	457.2	459	0.6	2.0
Attiro	**743**	**0.31**	**743**	743.0	743	0.0	0.5	**444**	**0.61**	**444**	450.4	474	10.3	0.7
LindenStrasse	**1054**	**1.34**	**1054**	1057.8	1090	10.7	5.9	**429**	**2.82**	**429**	431.7	445	4.6	12.7
SmallWorld	**4629**	202.87	**4629**	4660.5	4734	48.1	**51.4**	**1694**	117.54	**1694**	1694.0	1694	0.0	**58.6**
NetScience	**2102**	**9.07**	**2102**	2102.0	2102	0.0	52.9	**897**	**7.12**	**897**	901.0	927	8.8	112.3
USAir97	**10623**	377.83	**10623**	10697.2	10729	48.6	**269.7**	**3100**	1277.38	**3100**	3219.1	3405	105.9	**423.6**

Table 4.4: Results for real-world instances: Optimal objective value (Opt) and summary statistics for heuristic (minimum (min), mean (avg), maximum (max) and standard deviation (std)) with budget settings $B = 0.05n$ & $B = 0.1n$. Values compared are Opt and min, lower values are better (best in bold)

Synthetic instances

Results for our first set of synthetic networks are summarised in Table 4.5 as well as Figures 4.1 – 4.4 . Overall, the heuristic attains the best known upper bounds in 16.7% of the instances, yielding new upper bounds in 57.1% but falls short of the best UB in 26.2% of the instances (see Figure 4.1).

From Figures 4.2 and 4.3, we can observe that for the instances where the heuristic falls short, the quality of the solutions is still competitive in comparison to the best UB.

	B=0.05n								B=0.1n							
Graph	**LB**	**UB**	**t_exact**	**min**	**avg**	**max**	**std**	**t_heur**	**LB**	**UB**	**t_exact**	**min**	**avg**	**max**	**std**	**t_heur**
ba1(3)	**4275.0**	**4275**	330.19	**4275**	4275.0	4275	0.0	**8.5**	**3330**	**3330**	278.57	**3330**	3330.0	3330	0.0	**13.6**
ba1(6)	**4278.0**	**4278**	303.56	**4278**	4278.4	4282	1.2	**9.8**	**3390**	**3390**	476.99	**3390**	3391.0	3395	2.0	**14.1**
ba1(9)	**4193.0**	**4193**	169.48	**4193**	4193.0	4193	0.0	**7.7**	**3328**	**3328**	374.92	**3328**	3328.4	3332	1.2	**13.6**
ba2(3)	4384.2	**4461**	3600	4465	4465.0	4465	0.0	**20.9**	3716.0	**3987**	3600	4005	4005.0	4005	0.0	**46.1**
ba2(6)	**4369.0**	**4369**	562.95	4436	4453.4	4465	14.2	**19.3**	3717.5	**3916**	3600	3955	3955.9	3956	0.3	**44.5**
ba2(9)	4371.4	**4463**	3600	4465	4465.0	4465	0.0	**18.3**	3702.2	**3986**	3600	4004	4004.0	4004	0.0	**48.6**
er1(3)	2798.0	**2835**	3600	2842	2843.4	2845	1.2	**9.6**	2394.6	**2474**	3600	2535	2538.7	2549	5.0	**20.5**
er1(6)	2799.4	**2835**	3600	**2835**	2839.6	2848	5.7	**11.2**	2394.7	**2482**	3600	2485	2519.1	2540	12.8	**22.9**
er1(9)	**2814.0**	**2814**	1787.2	2847	2847.7	2850	0.9	**8.4**	2378.4	**2452**	3600	2528	2539.0	2548	6.0	**20.6**
er2(3)	15989.5	16955	3600	**16842**	16897.3	16909	19.9	**101.5**	12468.9	14886	3600	**14332**	14360.0	14385	14.0	**213.5**
er2(6)	16025.5	16930	3600	**16887**	16930.0	16960	28.6	**91.5**	12575.4	15052	3600	**14364**	14380.2	14403	11.9	**217.2**
er2(9)	15969.7	16954	3600	**16899**	16900.9	16908	3.4	**129.1**	12414.4	15038	3600	**14397**	14434.0	14488	24.1	**255.7**
gnm1(3)	15972.3	16771	3600	**16706**	16716.6	16740	9.3	**136.4**	12516.5	14730	3600	**14193**	14256.3	14296	31.2	**252.2**
gnm1(6)	16209.4	17062	3600	**16975**	16977.4	16987	3.5	**104.6**	12600.7	14658	3600	**14399**	14419.0	14446	13.4	**221.4**
gnm1(9)	16099.0	16958	3600	**16843**	16860.1	16886	15.9	**89.0**	12564.9	14803	3600	**14195**	14211.3	14221	12.0	**178.1**
gnm2(3)	34014.7	36803	3600	**35332**	35334.7	35341	4.1	**281.4**	25876.6	28978	3600	**28715**	28734.4	28764	21.0	**637.4**
gnm2(6)	33700.6	36445	3600	**35236**	35245.4	35252	6.0	**303.4**	25261.7	30635	3600	**28554**	28629.7	28704	48.5	**647.9**
gnm2(9)	33782.3	36641	3600	**35331**	35350.8	35388	18.8	**353.9**	25765.4	30805	3600	**28868**	28922.1	28956	25.0	**709.9**
gnm3(3)	36402.9	40229	3600	**39978**	39982.0	39983	2.0	**627.3**	28541.1	35847	3600	**35158**	35198.5	35238	25.0	**1539.6**
gnm3(6)	36557.1	40217	3600	**39848**	39876.1	39902	18.8	**681.6**	27307.3	35501	3600	**35000**	35079.6	35176	47.9	**1353.8**
gnm3(9)	36258.1	40176	3600	**39852**	39880.5	39896	14.8	**605.1**	28697.9	35704	3600	**34785**	34837.7	34935	50.7	**1323.1**

Table 4.5: Results for synthetic instances: Exact Lower and Upper bounds (LB, UB) and summary statistics for heuristic (minimum (min), mean (avg), maximum (max) and standard deviation (std)) for budget settings $B = (0.05n, 0.1n)$. Values compared are UB and min, lower values are better (best in bold)

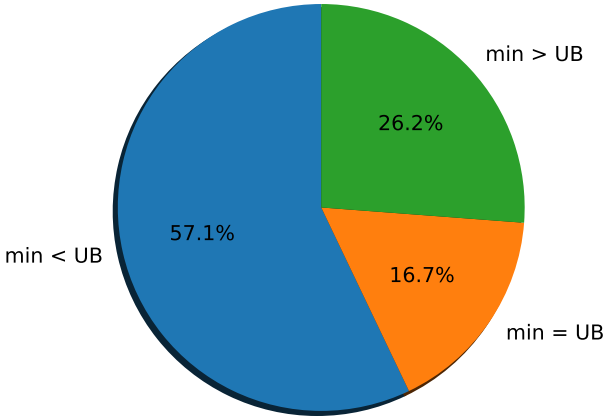
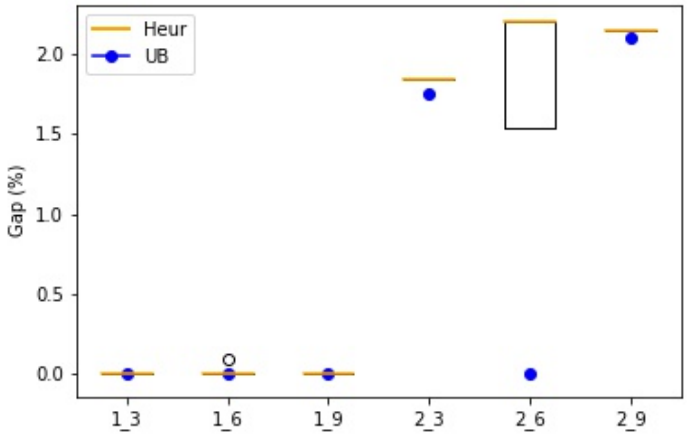
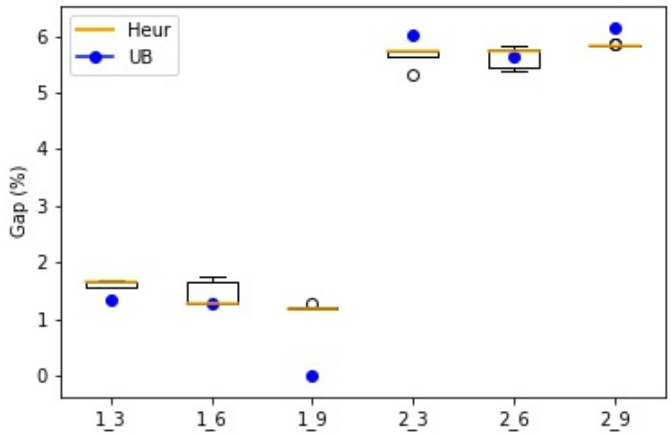


Figure 4.1: Summary results of heuristic algorithm compared with exact methods over synthetic instances

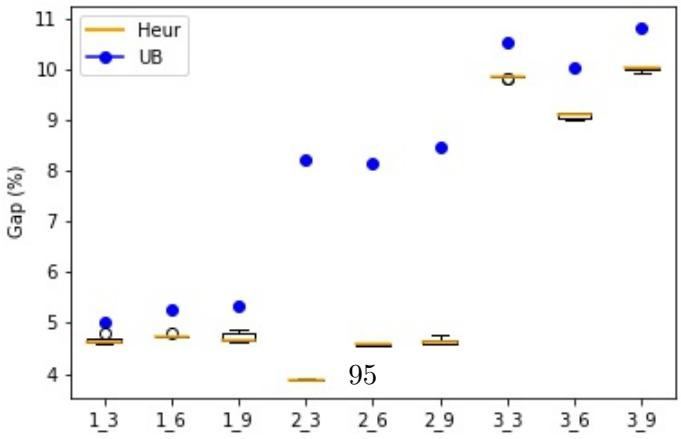
Analysing the performance of the heuristic across the synthetic network classes, we observed that the effectiveness of the heuristic framework is more pronounced in the less dense instances within each random network class. For example, for both Barabasi-Albert and Erdos-Renyi network classes, the heuristic yields new upper bounds in all the less-dense instances of both network classes (see results for **ba1** and **er2** instances in Table 4.5). However, as the edge density increases (graph characteristics are shown in Table 4.2), the heuristic falls short of the best known upper bounds matching only 1 out of the 6 **er1** instances and falling short in all **ba2** instances. This behaviour might be explained by the concept of "Structural Equivalence" wherein some of the most central nodes have overlapping neighbourhoods leading to redundancy of the solution set in which they occur (Borgatti 2006). Moreover, **ba2** and **er1**, being highly connected networks, are likely to suffer from the "Problem of Ties" which affects the performance of centrality-based algorithm on such topological structures (Ventresca & Aleman 2015). Our empirical investigation correlates with the above concepts. We



(a) BA instance

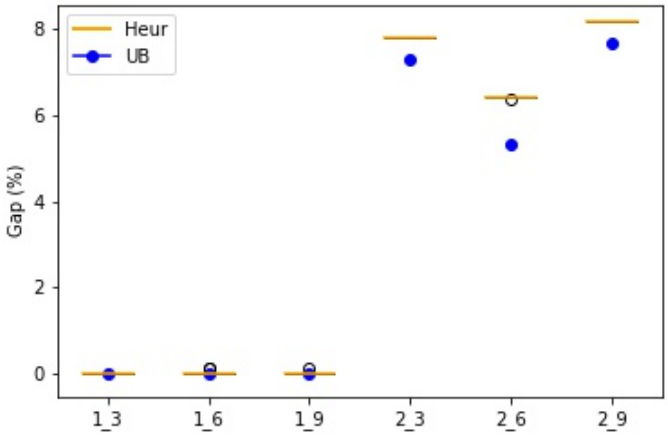


(b) ER Instance

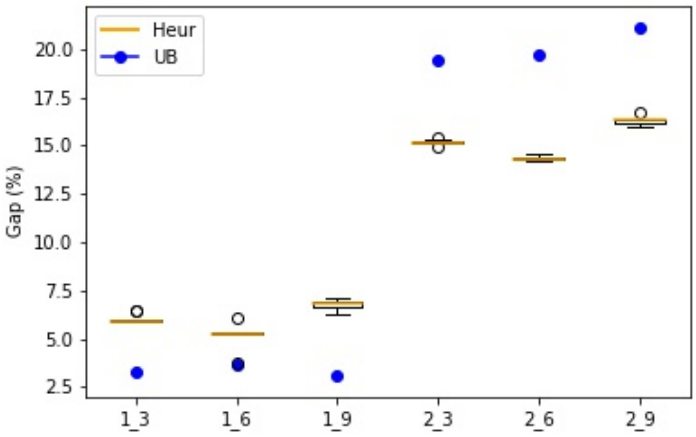


(c) GNM Instance

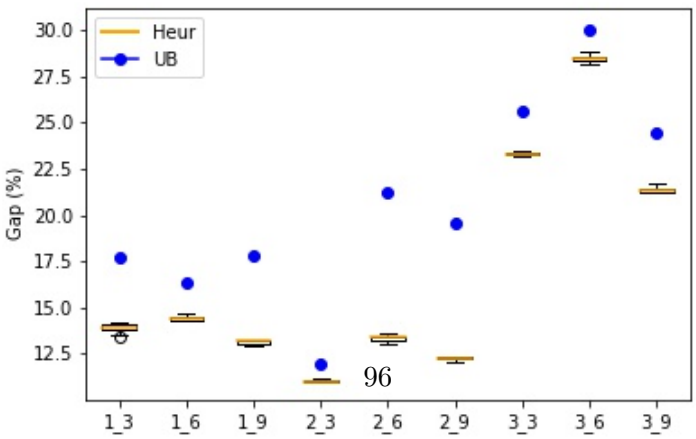
Figure 4.2: Comparison of gaps from best UB and Heuristic for the synthetic instances, B=0.05n



(a) BA instance



(b) ER Instance



(c) GNM Instance

Figure 4.3: Comparison of gaps from best UB and Heuristic for the synthetic instances, B=0.1n

observed the existence of multiple solutions with similar objective value differing only by one or two nodes. Thus, based on the structure of the centrality-based neighbourhood, the node swapping technique yielded little or no improvement as could be seen from the heights of the bars for **ba2** and **er1** in Figure 4.4. However, some improvement were observed for the less-dense Barabasi-Albert and Erdos-Renyi (**ba1, er2**) instances. This is also the case for the uniform random instances (**gnm1, gnm, gnm3**) which constitute the largest subset of the set of synthetic instances in terms of size and were also the most challenging synthetic network instances for the exact methods. The bars in Figure 4.4 represent percentage improvement calculated based on the objective function values realised before and after local search (InitialObj and FinalObj, respectively) averaged across all instances in each of the displayed synthetic graph type.

For a given problem instance, the percentage improvement is calculated as

$$\%improvement = 100 \cdot \frac{InitialObj - FinalObj}{InitialObj}\%$$

and averaged over all 10 runs of the instance. Across the three random network classes, the heuristics performed best in the uniform random network class, yielding new upper bounds in all 18 instances with all maximum objective values even less than the exact upper bounds. The instances in the uniform random graph class were the most challenging for the exact methods. Hence, achieving these new upper bounds shows the usefulness of the proposed algorithm in providing good solution for challenging problem instances. Overall we also observed that classes and instances of the random graphs that were challenging for the exact methods were also the most computationally intensive for the heuristic algorithm as can be seen in average computational times reported for the uniform random graphs on Table 4.5. Also, the impact of the local search procedure increases with increase in the budget setting B as would be expected since the size of the neighbourhood is directly proportional to B.

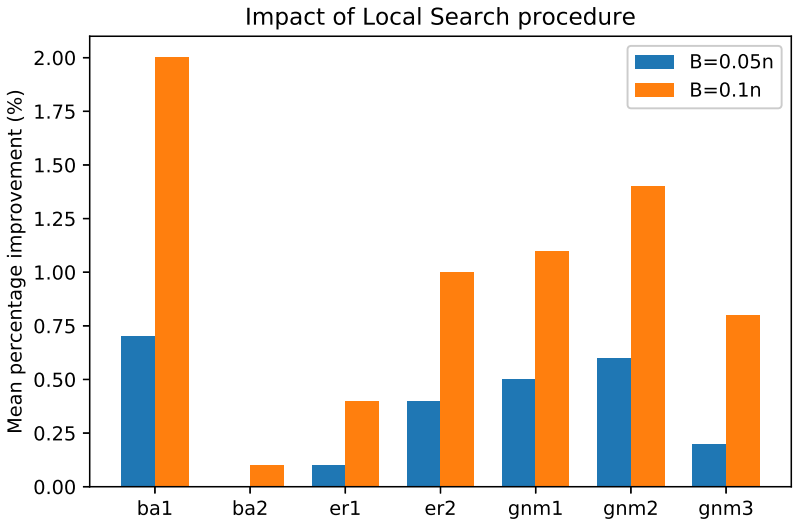


Figure 4.4: Variation of average percentage improvement in objective function values following the neighbourhood search procedure across different synthetic network types; budget settings $B = 0.05n$ & $0.1n$.

Benchmark instances

We extend our computational experiment to the set of benchmark synthetic graphs which have been used as test instances for most heuristic algorithms developed for the classical critical node detection problem. Due to the sparsity of these networks, we focused only on instances where the initial percentage k-distance connectivity (k-Conn%) is greater or equal to 20% (labelled as **FF250** and **WS250a** in Table 4.6). From Table 4.6, we observed that the heuristic realises the exact optimal objective value for 3 out 12 of the instances and yields new upper bounds in the remaining 9 instances. We also observed that as the budget increases especially for large dense graphs, the heuristic algorithm struggles to terminate within the specified time limit. In particular for **BA1000** and **ER1000**, the heuristic algorithm was unable to terminate within the specified time limit. These two instances were also the most challenging for the exact algorithm as seen from the gaps between the upper and lower bounds especially for **ER1000** for which an upper bound was only achieved after 8052 seconds. A possible way to enhance

the computational burden of the proposed heuristic for the larger instances might be to increase the probability of using a randomised node selection when repairing the partial offspring solution. However, the quality of the offspring solution might be affected in which case local search improvement can be employed with the gained time. Also an approximate evaluation of objective change might be employed during iterations of the greedy backbone crossover to reduce the computational burden of solution evaluation.

					Exact			Heur				
Graph	**n**	**m**	**k-Conn (%)**	**B**	**LB**	**UB**	**t_exact**	**min**	**avg**	**max**	**std**	**t_heur**
FF250	250	514	23.9	13	**1587.0**	**1587**	**11.44**	**1587**	1598.2	1601	5.6	16.7
BA250	250	1225	98.08	25	**13772.0**	**13772**	3160.27	**13722**	13788.25	13811	16.89	**137.6**
BA500	500	2475	95.18	50	**24847.0**	**24847**	3331.03	**24847**	24847	24847	0	**1104.7**
BA1000	1000	4975	84.97	100	16071.326	316735	3600	**59178**	60488.9	62487	909.7	3600.0
ER250	250	1190	94.04	25	17958.996	22288	3600	**19894**	19931.6	19970	19.89	**326.45**
ER500	500	2570	86.27	50	30845.690	79482	3600	**68062**	68129.2	68208	39.43	**3189.6**
ER1000	1000	5061	64.66	100	70494.315	221831	8052	**173538**	174326.1	175494	527.5	3600.0
WS250a	250	1246	52.9	70	1038.980	2319	3600	**2034**	2056.8	2093	17.0	**728.7**
WS250b	250	1250	79.65	25	14586.020	15223	3600	**15020**	15044.67	15086	16.55	**316.1**
WS500	500	2500	69.35	50	25907.377	53729	3600	**51460**	51567.2	51659	63.2	**3483.39**
GNM250	500	1250	96.17	25	18638.331	22711	3600	**20967**	20984.5	21013	17.0	**453.8**
GNM500	1000	2500	84.74	50	29461.785	78001	3600	**65775**	65892.9	65975	67.1	**2883.02**

Table 4.6: Results for benchmark synthetic instances: Exact Lower and Upper bounds (LB, UB) and summary statistics for heuristic (minimum (min), mean (avg), maximum (max) and standard deviation (std)). Budget settings for each instance are specified in column labelled B. Values compared are UB and min, lower values are better (best in bold)

4.5.4 Two-parent versus three-parent backbone crossover

We conclude the discussion on computational experiments by comparing the proposed three-parent backbone crossover to the usual two-parent approach. Recall that three initial feasible solutions are generated based on three centrality measures which are all used to generate offspring solutions (see Sections 4.3.3 & 4.3.4). We compare this to a two-parent approach in which we randomly select two out of the initial solutions as was

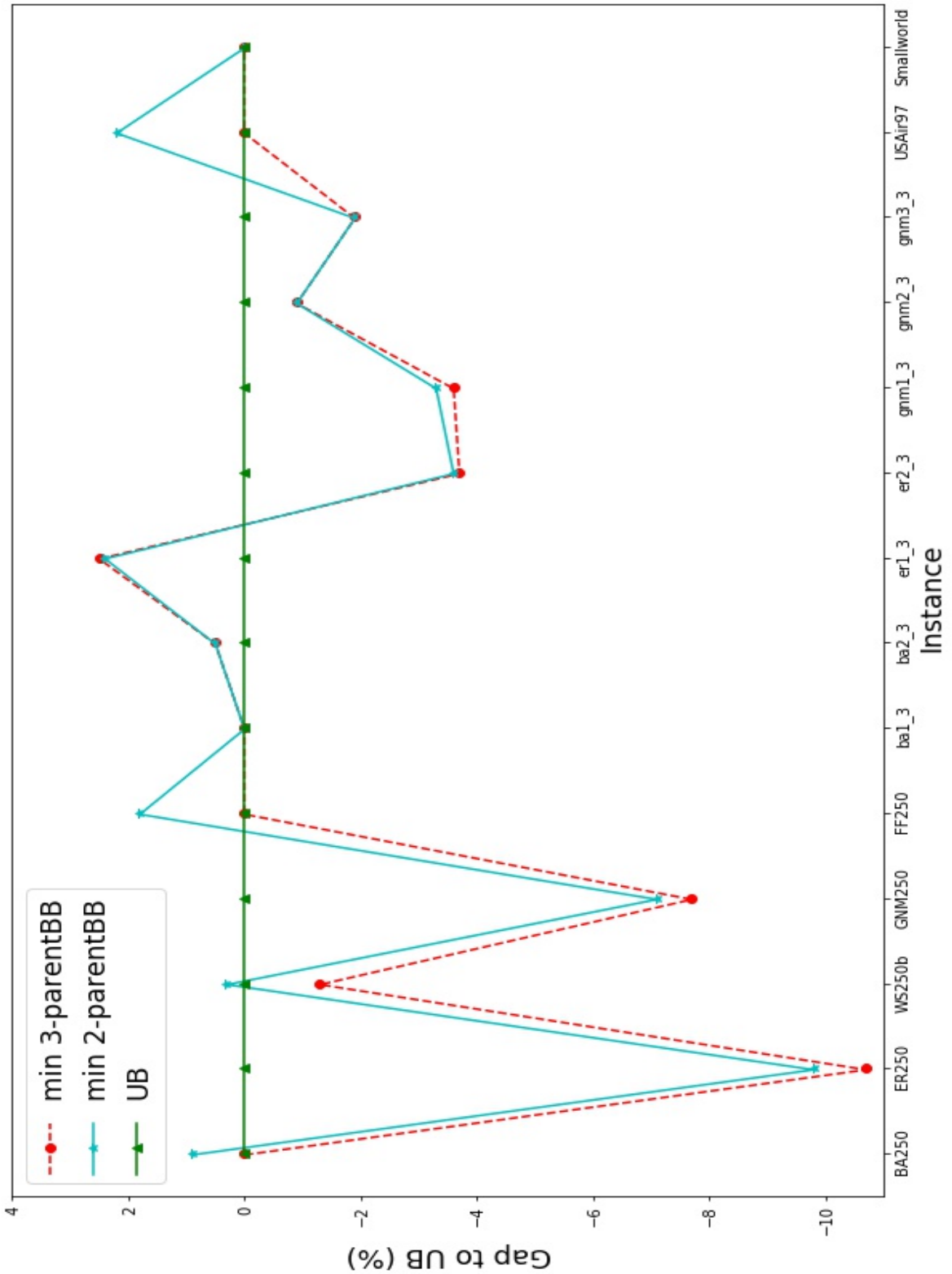


Figure 4.5: Comparison of crossover approaches: minimum objective values

done in Zhou et al. (2018). Using the two selected parent solutions, we construct an offspring solution using a combination of greedy and random backbone crossover.

Specifically, the initial offspring solution is repaired using the greedy approach 70% of the time and the random approach 30% of the time just as with the three-parent approach. The random approach selects from the corresponding *1-parent elements* or *0-parent elements* based on probabilities 0.6 and 0.4 respectively. Comparative performance of the proposed three-parent backbone crossover and its alternative two-parent in terms of the minimum objective value (min) and mean objective value (avg) are displayed in Figures 4.5 and 4.6 respectively. The x-axis represents the instances while the y-axis represents the percentage gap between the objective values (minimum values and mean values) realised by the heuristic and the exact upper bounds (UB). The percentage gaps are calculated as

$$\%gap = 100 \cdot \frac{obj - UB}{obj}\%$$

where obj is the minimum or mean objective value. A gap less than zero implies that the heuristic approach (three-parent or two-parent) realises a new upper bound for the given instance.

From Figures 4.5 - 4.6 and Table 4.7, we observe that out of the 14 tested instances, the three-parent approach obtains better minimum and mean objective values in 10 and 9 instances respectively. On the other hand, the two-parent approach obtains better minimum and mean objective values respectively in 1 instance and 2 instances. Both approaches perform alike obtaining the same minimum and mean objective values respectively for the smaller Barabasi-Albert instances (**ba1_3, ba2_3**) as well as the **Smallworld** instance. The advantage of the three-parent becomes more obvious as the instance size increases particular for the benchmark synthetic graphs. We observed that while the three-parent approach yields 3 new upper bounds and achieves the exact optimal for 2 of the 5 tested benchmark instances, the two-parent approach obtains minimum objective values which are worse than the exact upper bounds. Actual minimum and mean objective values along with run times for both three-parent and

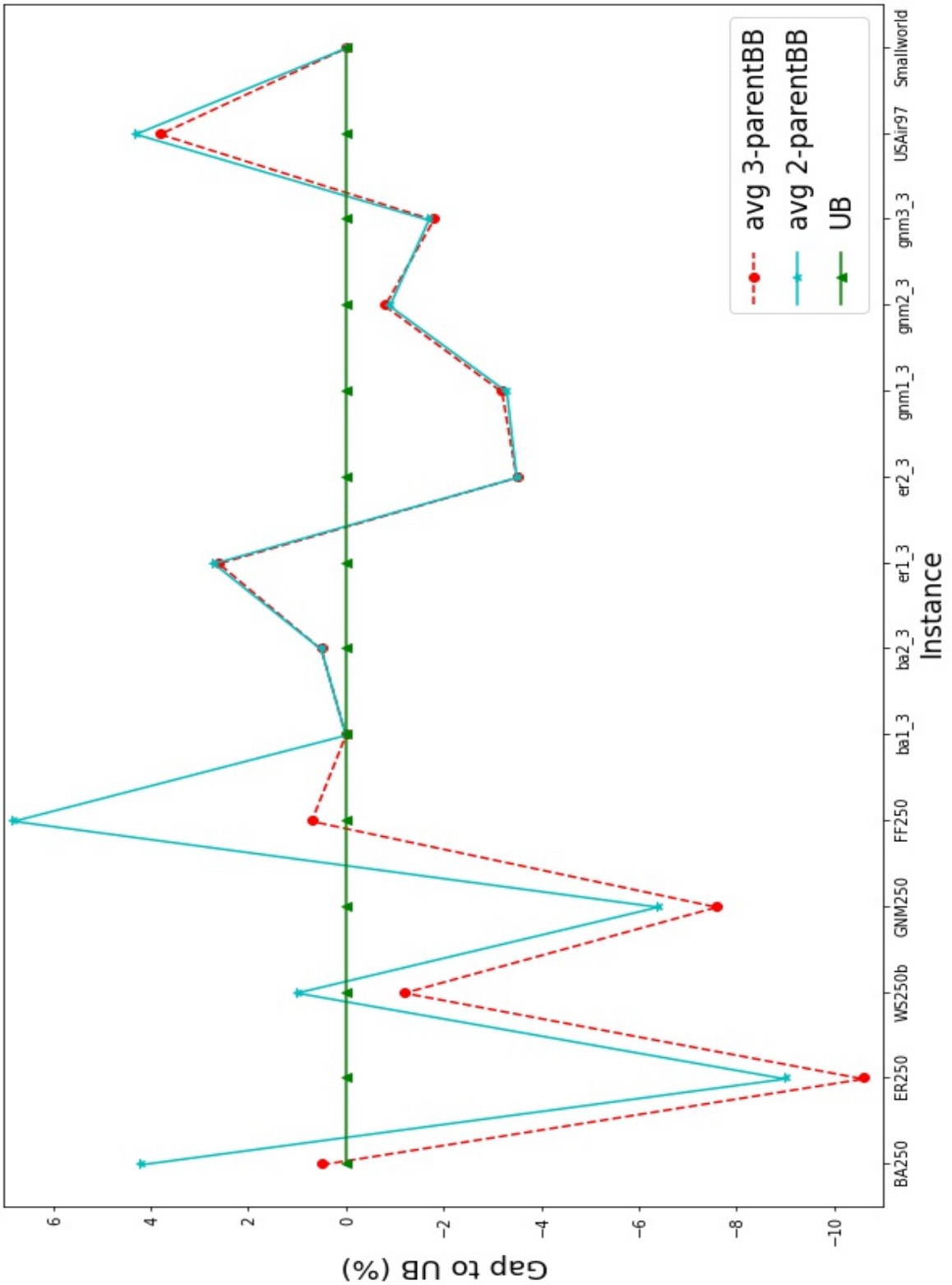


Figure 4.6: Comparison of crossover approaches: average objective values

					three-parent			two-parent		
Graph	**n**	**m**	**k-Conn (%)**	**UB**	**min**	**avg**	**t_heur**	**min**	**avg**	**t_heur**
SmallWorld	233	994	95.2	1694	**1694**	**1694**	58.6	**1694**	**1694**	**45.6**
USAir97	332	2126	84.8	3100	**3100**	**3219.1**	423.6	3168	3232.4	**293.5**
ba1_3	100	475	99.9	3330	**3330**	**3330.0**	**13.6**	**3330**	**3330.0**	13.8
ba2_3	100	900	100	3987	**4005**	**4005.0**	46.1	**4005**	**4005.0**	**42.9**
er1_3	80	470	100	2474	2535	**2538.7**	20.5	**2534**	2540.7	**17.75**
er2_3	200	1004	97.7	14886	**14332**	**14360.0**	213.5	14348	14370.4	**183.9**
gnm1_3	200	1000	97.9	14730	**14193**	14256.3	252.2	14244	**14249.8**	**196.3**
gnm2_3	300	1500	94.1	28978	**28715**	28734.4	637.4	28718	**28729.8**	**568.1**
gnm3_3	300	2000	99.6	35847	**35158**	**35198.5**	1539.6	35171	35235.0	**1353.5**
BA250	250	1225	98.08	13722	**13722**	**13788.3**	**137.6**	13844	14292.3	165.62
ER250	250	1190	94.04	22288	**19894**	**19931.6**	326.45	20107	20275.8	**263.99**
WS250b	250	1250	79.65	15223	**15020**	**15044.7**	316.1	15275	15370.3	**211.46**
GNM250	250	1250	96.17	22711	**20967**	**20984.5**	453.8	21088	21253.1	**431.09**
FF250	250	514	23.9	1587	**1587**	**1598.2**	16.7	1615	1694.2	**16.2**

Table 4.7: Comparison of the proposed heuristics using three-parent and two-parent backbone crossover approaches on a set of test instances, $B = 0.1n$ except for **FF250** where $B = 13$ as recorded from the source of the data. Values compared are minimum (*min*) and mean (*avg*) objective values obtained from both approaches as well as run times (*t_heur*), lower values are better (best in bold). The exact upper bounds are given in column labelled *UB*

two-parent backbone crossover can be seen in Table 4.7. Based on the tested instances, the three-parent approach performs better than the two-parent approach in terms of minimum and mean objective values however at higher cost of run times. This could be attributed to the higher number of iterations of the greedy procedure required to repair the initial offspring solution obtained from the common elements of all three parents. An approximate method to evaluate changes in objective value within the greedy procedure could be useful to improve the run times.

4.6 Concluding remarks

In this chapter, we considered a class of distance-based critical node detection problem. The proposed heuristic algorithm generates good solutions following a combination of greedy and randomized backbone-based crossover on initial feasible solutions. We also presented an improvement scheme that is derived from a centrality-based neighbourhood search. Extensive computational experiments on both real-world and synthetic graphs show the usefulness of the developed heuristics in generating good solutions when compared to exact solutions particularly for challenging problem instances. For the synthetic graphs which comprise of 54 instances, the heuristic achieves the optimal objective value or best known upper bounds on 17% of the instances within short run time (on average 20 times as fast as the exact approaches). Most importantly, amongst the challenging instances such as the uniform random graph were exact percentage optimality gaps were as high as 100%, the heuristic discovers new upper bounds on 57% of the instances within a maximum run time less than half of the time limit of 3600 seconds. These results indicate that the heuristic can be employed as a warm-start algorithm to provide good solution in small running time to kick start the exact algorithm. Such a strategy has the potential of speeding up the optimisation process of the exact approach and has been employed to solve difficult optimisation problems including a class of the critical node detection problem (see for example, Salemi & Buchanan (2020). Finally, the computational experiments provided some insights about the structural properties of

the dense classes of the Barabasi-Albert and Erdos-Renyi network where the heuristic fell short in terms of objective value. This is potentially useful for future development of heuristic algorithms for these and related network classes.

Chapter 5

Generalisation for edge-weighted distances and edge deletion

5.1 Introduction

In this chapter, we present a generalisation of the idea of the proposed path-based formulation to edge weighted graphs where distances are not limited to hop-based distances. This is motivated by real-world cases where edge weights represent costs such as construction cost, travel time, etc and as such vary from one edge to another. We also consider a related distance-based connectivity problem which we call the distance-based critical edge detection problem (DCEDP) in which the entities to be deleted are edges. We show how the model for the node-deletion version could be easily modified for the edge-deletion version as well as an alternative model for the distance-based critical edge detection problem.

5.2 Distance-based critical node detection problem for edge weighted graphs

We show how to generalise the integer programming models proposed in Chapter 3 for edge-weighted graphs. Let $G = (V, E)$ be a given undirected edge-weighted graph. For

every edge $(i,j) \in E$, we denote by w_{ij} a positive weight on (i,j) interpreted as the length of (i,j). For any pair of nodes i and j, the length of any given path between i and j is the sum of all edge weights along that path. Then the distance between i and j is defined as the length of a shortest path between i and j. We assume that all edge weights w_{ij} are positive integers which is not too restrictive for most real-world applications. However, where edge weights are rational numbers, the procedure proposed in Veremyev et al. (2015) can be used to modify rational weights. Using similar notations as in the unweighted case i.e formulation (3.19) -(3.26), we define $y_{ij}^{\ell} = 1$ if and only if there is a path of length at most ℓ in the induced graph G^R where $\ell \in \{1, \ldots, L\}$ and $L \leq (n-1).\max_{(i,j)\in E}\{w_{ij}\}$. The distance-based critical node detection problem for edge-weighted distances admits the following formulation:

DCNDP-PBML-gen

$$\text{minimise} \quad \sum_{i,j\in V: i<j} \left(f(1) y_{ij}^{1} + \sum_{\ell=2}^{L} f(\ell) \left(y_{ij}^{\ell} - y_{ij}^{\ell-1} \right) \right) \tag{5.1}$$

$$+ \sum_{i,j\in V: i<j} f(\infty) \left(1 - y_{ij}^{L}\right) \tag{5.2}$$

$$\text{s.t.} \quad y_{ij}^{\ell} = 0, \quad \forall (i,j) \in E,\ i<j,\ \ell \in \left\{1, \ldots, \min_{t\neq j:(i,t)\in E} \{w_{it}, w_{ij} - 1\}\right\} \tag{5.3}$$

$$y_{ij}^{\ell} = 0, \quad \forall (i,j) \notin E,\ i<j,\ \ell \in \left\{1, \ldots, \min_{t:(i,t)\in E} \{w_{it}\}\right\} \tag{5.4}$$

$$\sum_{r\in V(P)} x_r + y_{ij}^{d} \geq 1, \quad \forall P \in \mathcal{P}_L(i,j),\ i,j \in V,\ i<j \tag{5.5}$$

$$y_{ij}^{\ell} \leq y_{ij}^{\ell+1}, \quad \forall (i,j) \notin E,\ i<j,\ \ell \in \{2, \ldots, L-1\} \tag{5.6}$$

$$y_{ij}^{\ell} = y_{ij}^{(w_{ij})}, \quad \forall (i,j) \in E,\ i<j,\ \ell \in \{w_{ij}+1, \ldots, L\} \tag{5.7}$$

$$\sum_{i\in V} x_i \leq B \tag{5.8}$$

$$y_{ij}^{\ell} \in \{0,\, 1\}, \quad \forall (i,j) \in V,\ i<j,\ \ell \in \{1, \ldots, L\} \tag{5.9}$$

$$x_i \in \{0,\, 1\}, \quad \forall i \in V \tag{5.10}$$

Constraints (5.5)-(5.10) are based on the same ideas as those of constraints (3.21)-(3.26) in the path formulation for hop distance. Constraints (5.3)-(5.4) enhance the formulation

in a similar version as the shortest path-based enhancements described for the base formulation. Thus constraints (5.3)-(5.4) fix y_{ij}^{ℓ} variables to zero for all ℓ less than the minimum distance from node i to its neighbours t since the length of any shortest path from node i to any other node j must be greater than this minimum. In constraint (5.5), instead of the hop-based length of path P, we now consider the edge-weighted length of path P represented by d. As with the hop-based distance DCNDP, the non-redundant constraints in (5.5) are those corresponding to edges that is

$$x_i + x_j + y_{ij}^{(w_{ij})} \geq 1, \quad \forall (i,j) \in E,\ i < j$$

The rest are added dynamically through an appropriate separation routine. The separation routine follows a similar fashion as with the hop distance counterpart. For an integer solution, we construct a residual graph $G'(V', E')$ containing only nodes whose values in the current integer solution is equal to zero along with the edges incident on them. That is $V' = \{v \in V : \tilde{x}_v = 0\}$ and $E' = \{(u,v) \in E : u, v \in V'\}$. Then, using this new graph G', the separation routine entails generating a shortest path tree T_u rooted at each $u \in V'$. For every v in the tree T_u whose distance (d) from the root u is within the specified threshold L, the corresponding constraint (5.5) is checked for violation. That is, if $\tilde{y}_{uv}^{d} < 1, \quad \forall v \in T_u$, then the violated inequality $\sum_{i \in V(P_{uv})} x_i + y_{uv}^{d} \geq 1$ is added to the formulation.

Observe that for arbitrary edge weights, the size of formulation (5.1)–(5.10) becomes pseudopolynomial. To overcome this, we aggregate the y variables into a new set of continuous variable y_{ij} which is independent of ℓ. This new variable y_{ij} is interpreted as the value of the distance function between nodes i and j. Using these aggregated continuous y_{ij} variables, we arrive at an alternative formulation for the weighted DCNDP defined as follows:

DCNDP-PBM-gen

$$\text{minimise} \quad \sum_{i,j \in V: i<j} y_{ij} \tag{5.11}$$

$$\text{s.t.} \quad \sum_{r \in V(P)} f(d)x_r + y_{ij} \geq f(d), \quad \forall P \in \mathcal{P}_L(i,j),\ (i,j) \in V,\ i<j \tag{5.12}$$

$$\sum_{i \in V} x_i \leq B \tag{5.13}$$

$$x_i \in \{0,\ 1\}, \quad \forall i \in V \tag{5.14}$$

$$y_{ij} \geq 0, \quad \forall (i,j) \in V,\ i<j \tag{5.15}$$

Objective (5.11) minimises the connectivity of the input graph with respect to a specified distance-based connectivity function $f(.) \geq 0$. Formulation (5.11)–(5.15) above admits any non-negative non increasing distance function such as the first 3 classes of the DCNDP. This is captured in the definition of constraints (5.12) and the y_{ij} variables. Constraints (5.12) basically indicate that for any pair of nodes i and j connected by a path of length at most L, if none of the nodes along the paths connecting i and j is deleted, then the distance-based connectivity between i and j is at least $f(d)$ where d is the length of path P. Since the objective is a minimisation and the distance function is non-decreasing, we see that y_{ij} would be equal to $f(\bar{d})$ the value of the distance function corresponding to the length of shortest path between i and j. Thus the non-redundant inequalities in constraint set (5.12) are those of the shortest path between each pair of nodes which can be separated through generation of shortest path trees following the procedure discussed for the disaggregated path formulation.

It is easy to see that when the distance function $f(d)$ is defined as in equation (3.6), Constraints (5.12) reduces to

$$\sum_{r \in V(P)} x_r + y_{ij} \geq 1, \quad \forall P \in \mathcal{P}_L(i,j),\ (i,j) \in V,\ i<j$$

which are the path-based constraints for DCNDP-1 formulation (3.28)–(3.32). It can also be shown that formulation (5.11)–(5.15) is a valid formulation for the rest of the distance functions defined in Section 3.2 of Chapter 3. We illustrate this for two other DCNDP

objective namely: minimise the efficiency of the graph and minimise the sum of power functions of distances in the graph. Recall that the distance functions for these DCNDP objectives are defined respectively as $f(d) = d^{-1}$ and $f(d) = p^d, \quad 0 < p < 1$ if d less or equal to the specified threshold L and 0 elsewhere. By substituting the corresponding values of each of these distance functions into constraints (5.12), one realises that the y variables would take on the value of the distance function of the shortest path between any node pair if none of the nodes along that path is deleted. For the last two DCNDP classes whose objectives have a maximisation sense, the formulation (5.11)-(5.15), can easily be adapted to accommodate their distance functions. For example, for class 4 of the DCNDP whose objective maximises the Wiener index, we redefine the distance function similar to that presented by Hooshmand et al. (2020):

Class 4. *Maximise the generalised Wiener index or, equivalently, the characteristic path length of the graph:*

$$f(d) = \begin{cases} d, & \text{if } d \leq L \\ M, & \text{if } d > L \end{cases} \tag{5.16}$$

where M is a sufficiently large constant such that $M > L$. Then, objective (5.11) changes to maximisation sense and constraint (5.12) is modified to:

$$y_{ij} \leq d + M \sum_{r \in V(P)} x_r, \quad \forall P \in \mathcal{P}_L(i,j),\ (i,j) \in V,\ i < j \tag{5.17}$$

In addition, the y variables are bounded by M, that is:

$$y_{ij} \leq M\ \forall (i,j) \in V,\ i < j$$

The variables y_{ij} measure the shortest path length between nodes i and j in the residual graph. Thus, y_{ij} equal to the length of the shortest path between i and j in the residual graph if i and j are connected; otherwise, it is equal to M. The ideas behind constraints (5.17) is the same with those of constraints (5.12). They indicate that if none of the nodes along a candidate path P connecting a pair of nodes i and j is deleted, then, the length of shortest path between them is at least d which is the length of the path P. The separation of these constraints is exactly as discussed.

5.2.1 Comparing the two path based formulations

We now compare the two path formulations for the distance-based critical node detection problem. From a theoretical perspective, the disaggregated path formulation (5.1)-(5.10) seems to be a more tighter formulation (has better lower bound) than the aggregated version (that is, formulation (5.11)-(5.15)). To see this, observe that for every solution to the disaggregated model, there exist a solution to the aggregated version with the same objective value by setting $y_{ij} = \max(f(l)y_{ij}^l)$. However, in terms of scalabilty, the disaggregated formulation is not likely to scale well computationally for arbitrary edge weights. This is because for arbitrary edge weights, the disaggregated y variables grows in the order of $L \leq (n-1) * \max_{(i,j)\in E}\{w_{ij}\}$.

Edge weight			**B=0.05n**			**B=0.1n**		
Range	**diam**	**InitObj**	**FinObj**	**PBMt (s)**	**PBMLt (s)**	**FinObj**	**PBMt (s)**	**PBMLt (s)**
{1}	4	45.44%	9.28%	**238.6**	475.7	4.10%	**595.5**	1631
{1, 2}	8	23.44%	4.64%	**251.5**	599.2	2.01%	**822.9**	1045.2
{1, 2, 3}	12	16.01%	3.09%	**251.5**	747.9	1.34%	**716.5**	932.8
{1, 2, 3, 4}	16	12.18%	2.32%	**234.4**	925.5	1.01%	**786.4**	1822.7
{1, 2, 3, 4, 5}	20	9.83%	1.86%	**201.5**	1217.5	0.80%	**996.5**	1731.7
{1, 2, 3, 4, 5, 6}	24	8.28%	1.55%	**211**	1340.1	0.67%	**1040.5**	2452.6

Table 5.1: Comparing aggregated and disaggregated path-based models on weighted instances of the SmallWorld network using DCNDP-2 objective

We investigate the computational efficiency of the two path formulations using their respective models for the first two DCNDP classes DCNDP-1 and DCNDP-2. Recall that the objective in DCNDP-1 is to minimise the number for nodes connected by a path of length at most k and while DCNDP-2 minimises the efficiency. Comparison for the DCNDP-1 class is based on the larger real-world instances described in Chapter 3.5 using the hop-based distance models (unit edge weights). For DCNDP-2, our comparisons use the edge-weighted distance models on the smaller real-world network as well as the **SmallWorld** network. The **SmallWorld** network posseses characteristics of many real-world network having a small diameter of 4. We assign positive integer weights

on edges of the network based on their edge betweenness centralities (Brandes 2008). Following the same assumption made by Veremyev et al. (2015), we assign smaller edge weights to edges with large edge betweenness centrality values. Specifically, given the normalised edge betweenness b_{ij} of edge (i, j), we set $w_{ij} = min(b, \bar{b})$, where $\bar{b}$ is the nearest positive integer gotten from approximating the quotient of 0.1 and b_{ij} while b is some specified positive integer that provides an upper bound on the edge weights. For our experiment, we used values of b in the range $\{1, 2, 3, 4, 5, 6\}$ for the **SmallWorld** network where $b = 1$ is equivalent to the hop-based distance. We set $b = 6$ for the rest of the real-world networks. Tables 5.1 and 5.2 summarise the results of edge-weighted distance versions of both path-based models using DCNDP-2 objective while results of the hop-distance versions using DCNDP-1 objective are summarised in Table 5.3. The

					B=0.05n			**B=0.1n**		
Graph	**n**	**m**	**diam**	**InitObj**	**FinObj**	**PBMt (s)**	**PBMLt (s)**	**FinObj**	**PBMt (s)**	**PBMLt (s)**
Hi-tech	33	91	14	16.2%	13.0%	**0.89**	1.08	8.5%	**1.2**	2.57
Karate	34	78	11	21.8%	10.5%	**0.27**	0.3	2.7%	0.22	**0.19**
Mexican	35	117	16	13.6%	10.4%	**0.69**	1.93	5.7%	**0.58**	0.75
Chesapeake	39	170	16	12.0%	9.3%	**0.86**	2.43	4.8%	**0.47**	0.55
Lesmiserable	77	254	21	11.0%	3.2%	**4.87**	28.53	1.3%	**1.19**	4.56
Sawmill	36	62	12	23.7%	10.1%	**0.31**	0.41	5.3%	**0.57**	0.61
Dolphins	62	159	21	12.0%	5.5%	**2.93**	4.72	4.0%	**8.69**	23.5
Santafe	118	200	20	8.4%	0.6%	**2.33**	5.55	0.2%	**2.19**	18.63
Sanjuansur	75	155	24	10.5%	5.8%	**8.74**	39.95	2.6%	**5.87**	23.12
Attiro	59	128	22	11.8%	8.2%	**3.95**	16.75	4.3%	**3.23**	13.81
Smallworld	233	994	24	8.3%	1.6%	**211**	1340.12	0.7%	**1040.45**	2452.6

Table 5.2: Comparing aggregated and disaggregated path-based models on weighted real-world instances of DCNDP-2 where edge weights are in the range $\{1, 2, 3, 4, 5, 6\}$

columns labelled **PBMt(s)** contain the running time of the aggregated path formulation while the columns labelled **PBMLt(s)** show the running time for the disaggregated path formulation. The objective values before and after node deletions are respectively displayed in columns labelled **InitObj** and **FinObj**. For both budget settings of 5% and 10% of input network size, one can see from Table 5.1 that the disaggregated model is

computationally more expensive than the aggregated model based on the **SmallWorld** network. This is indeed true for the different range of the edge weights, as well as for other network instances (see Table 5.2). Similarly with DCNDP-1 objective and unit edge weights, we see the same pattern across the different network instances and budget settings as shown in Table 5.3. Therefore, we can conclude that the aggregated path model (formulation (5.11)–(5.15)) is computationally more competitive than the disaggregated version (formulation (5.1)–(5.10)). This supports our motivation for the reformulation presented in Chapter 3 for the first class of the distance-based critical node detection problem and the generalisations to edge-weighted distance presented in this chapter.

Graph	**n**	**m**	**diam**	**InitObj**	**B**	**FinObj**	**PBMt (s)**	**PBMLt (s)**
USAir97	332	2126	6	84.8%	0.1n	5.64%	**1277.38**	2375.76
					0.05n	19.33%	**377.83**	526.39
					0.03n	39.35%	**692.04**	1054.04
					0.02n	49.57%	**582.16**	925.6
					0.01n	63.90%	**240.67**	474.64
SmallWorld	233	994	4	95.2%	0.1n	6.27%	**117.54**	156.78
					0.05n	17.13%	202.87	**170.95**
					0.03n	23.41%	**18.46**	25.71
					0.015n	40.56%	**41.17**	77.65
NetScience	379	914	17	13.3%	0.03n	4.32%	**5.11**	14.76
					0.01n	8.43%	**4.52**	7.8
LindenStrasse	232	303	13	12.1%	0.4n	4.70%	**1.17**	2.03
					0.03n	6.15%	**1.82**	3.32
					0.015n	8.32%	**1.25**	2.11

Table 5.3: Comparing aggregated and disaggregated path-based models on unweighted real-world instances of DCNDP-1

5.3 Distance-based critical edge detection problem

We consider two formulations for the edge-deletion version of the distance-based critical node detection problem which we refer to as the distance-based critical edge detection problem (DCEDP). The first is a modification of the node deletion case. Let's consider for simplicity an undirected graph $G = (V, E)$, we can construct an auxiliary graph $G' = (V + V', E')$ as follows:

- For each edge $(i, j) \in E$, add a new node k and replace $(i, j) \in E$ with (i, k) and (k, j). Thus the new graph G' has $n + m$ nodes and $2m$ edges.
- For hop-based distances, since the number of edges in the auxiliary graph is twice the number of edges in the input graph, the distance threshold would be modified as well to $2L$.
- For edge-weighted distances, the weight w_{ij} of edge $(i, j) \in E$ would be propagated equally to the corresponding edges of the auxiliary graph. Thus, the weights for edges (i, k) and (k, j) would be w_{ij} and the distance threshold would be modified to $2L$.
- For non-unit edge deletion costs, the deletion cost c_{ij} for edge $(i, j) \in E$ is assigned to the corresponding new node k that is $c_k = c_{ij}$.

Let us partition the nodes in the auxiliary graph G' into two disjoint sets V and V'. Let set V be the set of all nodes present in the initial input graph G while V' be the set of all new nodes added for each edge of G. Observe that deletion of nodes $v' \in V'$ from the auxiliary graph G' is equivalent to deletion of the corresponding edge of G. Thus, the distance-based critical edge detection problem on the input graph G is equivalent to the distance-based critical node detection problem on the constructed auxiliary graph G' formulated as follows:

DCEDP-mod

$$\text{minimise} \quad \sum_{i,j \in V : i<j} y_{ij} \tag{5.18}$$

$$\text{s.t.} \quad \sum_{r \in V'(P)} f(d) x_r + y_{ij} \geq f(d), \quad \forall P \in \mathcal{P}_L(i,j),\ i,j \in V,\ i<j \tag{5.19}$$

$$\sum_{i \in V'} c_i(x_i) \leq B \tag{5.20}$$

$$x_i = 0, \quad y_{ij} \geq 0, \quad \forall (i,j) \in V,\ i<j \tag{5.21}$$

$$x_i \in \{0,\, 1\}, \quad \forall i \in V' \tag{5.22}$$

The construction of the auxiliary graph doubles the graph size. However, we can reduce the variable and constraint size by fixing the x variables of the original nodes to zero and defining y_{ij} variables for only node pairs (i,j) of the original graph as in constraint (5.21). We end the section by presenting an alternative formulation for the edge-weighted distance-based critical edge detection problem which uses the input graph directly. Given an edge weighted graph $G = (V, E)$ with positive integer edge weights w_{ij} and edge deletion costs c_{ij} specified for every edge $(i,j) \in E$. We define a new set of variables for edges as follows:

$$x_{ij} = \begin{cases} 1, & \text{if edge } (i,j) \text{ is deleted,} \\ 0, & \text{otherwise.} \end{cases} \tag{5.23}$$

We keep the aggregated continuous variables y_{ij} as defined for the node deletion version. The distance-based critical edge detection problem on the input graph G admits the formulation:

DCEDP

$$\text{minimise} \quad \sum_{i,j \in V: i<j} y_{ij} \tag{5.24}$$

$$\text{s.t.} \quad \sum_{(r,t) \in E(P)} f(d) x_{rt} + y_{ij} \geq f(d), \quad \forall P \in \mathcal{P}_L(i,j),\ i,j \in V,\ i<j \tag{5.25}$$

$$\sum_{i,j \in E: i<j} c_{ij} x_{ij} \leq B \tag{5.26}$$

$$x_{ij} \in \{0,1\}, \quad \forall (i,j) \in E \tag{5.27}$$

$$y_{ij} \geq 0, \quad \forall (i,j) \in V,\ i<j \tag{5.28}$$

where objective function (5.24) minimises some distance-based connectivity function as defined by $f(.)$. Constraints (5.25) are the equivalent path constraints which state that to disconnect any node pair i and j, at least one edge along each qualifying path (that is, paths whose lengths are within the specified threshold L) connecting i and j must be deleted. Constraints (5.26) specify that the total edge deletion cost must not exceed the available budget while constraints (5.27)– (5.28) specify the domain of the decision variables x and y.

5.4 Concluding remarks

In this chapter, we presented generalisations of our models and algorithm proposed for the distance-based critical node detection problem to edge weighted graphs. Two path-based formulations were considered. The first which is pseudopolynomial in variable size for arbitrary edge weights is theoretically tighter. However, computationally, it is not as efficient as the second formulation which uses an aggregated version of the variables to overcome the computationally intractability posed by the dependence on the edge weights. We also considered the edge deletion version of the distance-based critical node detection problem which we termed the distance-based critical edge detection problem. We showed how the aggregated formulation proposed for the node deletion version can be modified to solve the edge deletion problem on an auxiliary graph of the input graph.

Finally, an alternative formulation that does not require modification of the input graph was presented.

Chapter 6

Conclusion and Future Research

In this thesis, we studied node and edge deletion problems in networks. We began by introducing a class of node and edge deletion problems that is concerned with the overarching property of network connectivity namely the critical node detection problem. The importance of connectivity objective metric in the study of the critical node detection problem as well as in other problems involving node and/or edge removals was presented. The various variants of the critical node detection problem, their applications and related problems as studied in literature were discussed. In addition, we discussed and illustrated some of the limitations of the variants of the critical node detection problem whose goal is network fragmentation. On account of these limitations, we briefly introduced the distance-based variant of the critical node detection problem which became the focus of the remainder of the thesis.

In Chapter 3, we formally defined the distance-based critical node detection problem along with the assumptions of the distance-based connectivity functions. We presented definitions of five existing distance-based connectivity functions and a compact integer programming formulation proposed in Veremyev et al. (2015). This compact integer programming model served as the base model for our study. Following this, we presented a new path-based model which like the base compact model is valid for all the five distance-based connectivity functions and any other functions satisfying the assumptions. The new integer programming model uses the same sets of decision variables as the

base model but its constraint set comprises of so-called lazy constraints. Inspired by the structure of the first distance-based connectivity function, we also present a reformulation of the path-based model by redefining the set of connectivity variables to one with fewer variables. Although the reformulation was devised with the first class of distance-based critical node detection problem in mind, we showed that it can be generalised to the other distance-based connectivity functions. The main assumptions of the models presented in Chapter 3 is that the input graph is unweighted, hence the distances considered are hop distances. Following this, we presented an algorithm for the separation problem associated with the lazy constraints in the new path-based models. The separation algorithm is based on modified breadth-first-search tree generation and as such explores the assumption of hop-based distances. As part of the discussion on the proposed algorithmic framework, we propose valid inequalities and a primal heuristic. The chapter ends with discussions on extensive computational experiments on both real-world and synthetic graphs. This includes a comparison of the proposed path-based models and the base compact model for two classes of the distance-based critical node detection problem where results show the computational competitiveness of our path-based models over the base model. We also investigated how the distance-based connectivity functions influence the the ease of solving the critical node detection problem for different network topology and size. The computational experiments compared the first 3 distance-based connectivity objective functions and the classical fragmentation-based pairwise connectivity objective. The results showed that the distance-based connectivity metrics were more computationally favorable for edge-dense graphs with small diameter with the first distance-based connectivity objective being the easiest to solve.
The influence of the proposed valid inequalities and primal heuristic on computational time and optimality gaps was also investigated. The distance-based critical node detection problem being an $\mathcal{NP}$-complete problem, exact methods are only able to solve small-medium size instances in reasonable time.

In consideration of the computational limitations of the exact approaches, Chapter 4 considered the distance-based critical node detection problem from a heuristic perspective.

The general framework of the proposed heuristic was presented. The framework combines a double backbone crossover and a neighbourhood search procedure. The double backbone crossover procedure generates offspring solution from initial solutions constructed based on defined centrality measures. The neighbourhood search procedure uses a two-phase node swap proposed by Zhou et al. (2018) on a reduced neighbourhood constructed based on centrality metrics of the nodes. Computational experiments on real-world and synthetics instances used for the exact methods in Chapter 3 as well as on larger benchmark synthetic instances were also presented. The computational results show that the proposed heuristic matches the optimal solutions in all the real-world instances and 19% of the synthetic instances. Moreover, the heuristic achieves new upper bounds in reasonable time for 61% of the synthetic network instances which were unsolved by the exact approaches. We also showed how the heuristic framework can be extended to other classes of the distance-based critical node detection problem as well insights on some topological structures where the heuristic failed to match the exact solution.

Finally, in Chapter 5, we considered the distance-based critical node detection problem on edge-weighted graphs. We extended the ideas of the two path-based formulations to edge-weighted distances where we assumed that edge weights are positive integers. This assumption is not limiting since this is the case in many combinatorial optimisation problems and as shown in Veremyev et al. (2015), fractional edge weights could be transformed to integer weights. We also considered edge deletion variant which we term the distance-based critical edge detection problem. We explored certain transformations of the input graph under which solving the distance-based critical node detection problem on the transformed graph is equivalent to solving the edge deletion version of the problem. Since the transformation doubles the size of the graph even though the variable size is kept the same by variable fixing, we also presented an alternative integer programming formulation that does not require any transformation.

The results gathered from this study provide useful tools to model and solve node and edge deletion problems particularly as it concerns distance-based connectivity property in the node-deleted and/or edge-deleted subgraph. Nevertheless, there are undoubtedly

directions for future research that would provide additional insights to solve real-life problems involving node and edge deletions more efficiently.

The underlying assumption of the distance-based critical node detection problem presented in this thesis is that the decision maker knows the required budget on total cost of deleting the critical nodes or edges. However, this is not always the case. Sometimes, the decision maker wants to uncover the extent of vulnerability of the network to different levels of damage (node or edge deletions). This would provide some insights as to how much cost he needs to incur to protect his infrastructure to mitigate such vulnerability. In this sense, the relevant problem becomes the β-connectivity distance-based critical node detection problem where $\beta,\ 0 < \beta < 1$ is a bound on the proportion of original distance-based connectivity left after node deletion. A general formulation for hop-based distances can be written as:

$$\text{minimise} \quad \sum_{i \in V} c_i x_i \tag{6.1}$$

$$\text{s.t.} \quad \sum_{r \in V(P)} x_r + y_{ij}^{|P|} \geq 1, \quad \forall P \in \mathcal{P}_L(i,j),\ i,j \in V,\ i < j \tag{6.2}$$

$$y_{ij}^{l-1} \leq y_{ij}^{l}, \quad \forall (i,j) \in V,\ i < j,\ l \in \{2, \ldots, L\} \tag{6.3}$$

$$y_{ij}^{l} = y_{ij}^{1}, \quad \forall (i,j) \in E,\ i < j,\ l \in \{2, \ldots, L\} \tag{6.4}$$

$$\sum_{i,j \in V : i<j} \left(f(1) y_{ij}^{1} + \sum_{l=2}^{L} f(l) \left(y_{ij}^{l} - y_{ij}^{l-1} \right) \right) \leq \beta \mathcal{N} \tag{6.5}$$

$$y_{ij}^{l} \in \{0,\ 1\}, \quad \forall (i,j) \in V,\ i < j,\ l \in \{1, \ldots, L\} \tag{6.6}$$

$$x_i \in \{0,\ 1\}, \quad \forall i \in V \tag{6.7}$$

where objective function 6.1 minimises the total deletion cost. Constraints 6.2–6.4 are same with the ones in the path-based formulation 3.19–3.26 (**DCNP-PBML**) proposed in this thesis. Constraint 6.5 bounds the distance-based connectivity in the node-deleted subgraph to some specified proportion β of the initial distance-based connectivity of the input graph. The distance-based connectivity of the input graph $\mathcal{N}$ depends on the distance-based connectivity functions. For example, with respect to the first distance-based

connectivity function, $\mathcal{N}$ would be the total number of node pairs connected by a path of distance at most k. Constraints 6.6–6.7 specify the domain of the decision variables which are as defined in Chapter 3 of this thesis. Moreover, the β-connectivity distance-based critical node detection problem as well as the models considered in this thesis can easily be reformulated to address more realistic cases. In particular, cases where the decision maker is interested in assessing the degradation in the distance-based connectivity for only a specified set of commodities (origin-destination pairs) and not all node pairs in the network.

The heuristic proposed in this thesis leverages the effectiveness of a memetic-type algorithm in particular, the double backbone crossover and the neighbourhood search. This is evidenced by its success in achieving in reasonable time the optimal and near-optimal objective values for the medium sized instances as well as new upper bounds for the challenging instances. However, as the instance size increases, the cost of evaluating solutions and hence changes in objective values resulting from node swaps becomes significant. Devising a more efficient evaluation function to overcome this bottle neck is indeed a direction for future research. One interesting area worth exploring is to use machine learning algorithm to approximate the objective value for a given solution or solution change. In recent times, machine learning has been employed to solve difficult graph combinatorial optimisation problems such as the TSP and set covering problem (Khalil et al. 2017, Bengio et al. 2021). The first approach to incorporate machine learning to the proposed heuristics is to use the trained machine learning algorithm to choose the most promising node to add to the current offspring solution within the greedy procedure of the backbone crossover (Section 4.3.4). Also, in the neighbourhood search, the learned model can also be used to select the nodes to swap. Since these two decisions are repeated within the heuristic framework, the run time of the heuristic would be significantly reduced given that the learning phase of the machine learning algorithm is done offline. This would also enable the extension of the heuristic to a proper memetic algorithm in which a larger population of feasible solutions is constructed leading to more offspring solutions generated from multiple triples of parent solutions.

Each of these offspring solutions can then be improved by the local search and used as parent solutions for further backbone crossovers until termination criteria is reached. This would potentially lead to improved solutions for larger instances. Following on from this, the computationally enhanced heuristic framework can be extended to other classes of the distance-based critical node detection problem and related problems.

The heuristic framework proposed in this thesis uses a heuristic approach to construct initial population. Indeed, it has been proved that using a heuristic approach for construction of initial solutions yields better solutions than using a random approach (Baker & Ayechew 2003). However, our computational experiments suggest that for highly connected networks, the inherent issue of structural equivalence affects performance of the centrality-based heuristic. Hence, future heuristic approaches should explore ways of initialising parent population such that the neighbourhood of the solutions do not overlap too much. A plausible approach might be to hybridise heuristic and random approaches in the solution initialisation.

Finally, it would be interesting to use the underlying ideas of the models and algorithms described in Chapters 3–5 to solve practical problems. For example, using class 3 of the distance-based critical node detection problem (that is, distance connectivity function (3.3)), the path-based models could be adapted to solve contagion control problems. If we take the susceptible-infected (SI) disease spread model, the set of nodes V would then be split into two subsets namely infected nodes (I) and susceptible nodes (S). Given the transmission probability p ($0 < p < 1$) of a contagion from a node $i \in I$ to a neighboring node $j \in S$, then for non-adjacent nodes $i \in I$ and $j \in S$ the transmission probability can be estimated from the power function (p^d) of DCNDP-3. As this probability decreases with distance d, it therefore implies that susceptible nodes that are sufficiently far from infected nodes (that is, a large distance d) are less likely to contact the infection. Thus, a strategy for minimising the spread of such kind of contagion within a social or population network can be achieved by solving a modified version of the hop-constrained path-based model for DCNDP-3 proposed in Chapter (3). That is, we seek nodes (members of the network) whose removal (isolation or vaccination)

would maximally reduce the propensity of spread of the contagion (e.g virus). The modification is as follows: Instead of considering every node pair $i, j \in V$ in the objective function and path-based constraints, we only consider node pairs $i, j : i \in I\ ,\ j \in S$. The resultant aggregated model is given by formulation (6.8)–(6.12). Consequently, the separation algorithm for the path-based constraints (Algorithm 1) can be slightly modified by further restricting the set of candidate root nodes R_c to only infected nodes, that is, $R_c = R_c \cap I$. Furthermore, in checking for violations of the path-based constraints in line 17 of Algorithm 1, we restrict our check to susceptible nodes $t \in S$ at the corresponding tree depth. So that for every BFS tree generated, the root node r of each tree is an infected node (that is, $r \in I$) and the terminal node t along a path in the tree for which we check path constraint violation must be a susceptible node (that is, $t \in S$).

DCNDP-PBM-App

$$\text{minimise} \quad \sum_{i \in I\ ,\ j \in S} y_{ij} \tag{6.8}$$

$$\text{s.t.} \quad \sum_{r \in V(P)} f(d)\, x_r + y_{ij} \geq f(d), \quad \forall\ P \in \mathcal{P}_L(i,j),\ i \in I\ ,\ j \in S \tag{6.9}$$

$$\sum_{i \in V} x_i \leq B \tag{6.10}$$

$$x_i \in \{0,\, 1\}\,, \quad \forall\ i \in V \tag{6.11}$$

$$y_{ij} \geq 0, \quad \forall\ i \in I\ ,\ j \in S \tag{6.12}$$

We refer the interested reader to Nandi & Medal (2016) and Lalou et al. (2018) for related applications of the critical node detection problem to contagion control.

Another application problem where the models and algorithms proposed in this thesis can be adapted to solve is in assessing the vulnerability of a network a transportation network. One of the network instances used for the computational experiments described in Chapters (3)– (5) is the USAir97 network which is an air transportation network of United States air flights. Indeed, the computational experiments indicate that the first class of the DCNDP which minimises the number of node pairs connected by hop

distance less than or equal to 3 is a more appropriate model for this network. This is intuitive since many passengers would not expect an air trip from their origin location to destination to be more than 3 stops. A model for the traditional critical node detection problem has been used to assess the vulnerability of a road network to arc failures, where the existence of a path between two points in the road network is the metric used to assess vulnerability (Matisziw & Murray 2009). The models and algorithms proposed in this thesis can be used along with the distance function of DCNDP-1 or DCNDP-4 to assess the vulnerability of supply chain networks to node or edge failures. Indeed, the edge-weighted version considered in Chapter (5) would be a more appropriate model given that there are varied cost (distance or time) of flow of materials from one source location (production or storage facility) to a demand location. With a suitable estimation of these costs, and specification of a set of source-destination (s, t) points, one can apply the algorithm ideas for the weighted version of DCNDP or DCNEP to identify nodes or edges in the network whose loss or damage will maximise the cost (e.g. distance) required to move materials between (s, t) points. The modifications proposed for the contagion control application also would apply to this case, since we do not need to consider all node pairs.

Bibliography

Addis, B., Aringhieri, R., Grosso, A. & Hosteins, P. (2016), 'Hybrid constructive heuristics for the critical node problem', *Annals of Operations Research* **238**(1-2), 637–649.

Addis, B., Di Summa, M. & Grosso, A. (2013), 'Identifying critical nodes in undirected graphs: Complexity results and polynomial algorithms for the case of bounded treewidth', *Discrete Applied Mathematics* **161**(16-17), 2349–2360.

Aickelin, U. & Clark, A. (2011), 'Heuristic optimisation', *Journal of the Operational Research Society* **62**(2), 251–252.

Albert, R., Jeong, H. & Barabási, A.-L. (2000), 'Error and attack tolerance of complex networks', *nature* **406**(6794), 378.

Alozie, G. U., Arulselvan, A., Akartunalı, K. & Pasiliao Jr, E. L. (2021), 'Efficient methods for the distance-based critical node detection problem in complex networks', *Computers & Operations Research* **131**, 105254.

Aringhieri, R., Grosso, A., Hosteins, P. & Scatamacchia, R. (2016*a*), 'A general evolutionary framework for different classes of critical node problems', *Engineering Applications of Artificial Intelligence* **55**, 128–145.

Aringhieri, R., Grosso, A., Hosteins, P. & Scatamacchia, R. (2016*b*), 'Local search metaheuristics for the critical node problem', *Networks* **67**(3), 209–221.

Aringhieri, R., Grosso, A., Hosteins, P. & Scatamacchia, R. (2019), 'Polynomial and pseudo-polynomial time algorithms for different classes of the distance critical node problem', *Discrete Applied Mathematics* **253**, 103–121.

Arulselvan, A., Commander, C. W., Elefteriadou, L. & Pardalos, P. M. (2009), 'Detecting critical nodes in sparse graphs', *Computers & Operations Research* **36**(7), 2193–2200.

Arulselvan, A., Commander, C. W., Pardalos, P. M. & Shylo, O. (2007), 'Managing network risk via critical node identification', *Risk management in telecommunication networks, Springer* .

Arulselvan, A., Commander, C. W., Shylo, O. & Pardalos, P. M. (2011), Cardinality-constrained critical node detection problem, *in* 'Performance models and risk management in communications systems', Springer, pp. 79–91.

Baird, D. & Ulanowicz, R. E. (1989), 'The seasonal dynamics of the chesapeake bay ecosystem', *Ecological monographs* **59**(4), 329–364.

Baker, B. M. & Ayechew, M. (2003), 'A genetic algorithm for the vehicle routing problem', *Computers & Operations Research* **30**(5), 787–800.

Balakrishnan, A. & Altinkemer, K. (1992), 'Using a hop-constrained model to generate alternative communication network design', *ORSA Journal on Computing* **4**(2), 192–205.

Barabási, A.-L. & Albert, R. (1999), 'Emergence of scaling in random networks', *science* **286**(5439), 509–512.

Batagelj, V. & Mrvar, A. (2006), 'Pajek datasets', `http://vlado.fmf.uni-lj.si/pub/networks/data/`. Last accessed: 02-11-2018.

Bayrak, H. & Bailey, M. D. (2008), 'Shortest path network interdiction with asymmetric information', *Networks: An International Journal* **52**(3), 133–140.

Bengio, Y., Lodi, A. & Prouvost, A. (2021), 'Machine learning for combinatorial optimization: a methodological tour d'horizon', *European Journal of Operational Research* **290**(2), 405–421.

Boginski, V. & Commander, C. W. (2009), Identifying critical nodes in protein-protein interaction networks, *in* 'Clustering challenges in biological networks', World Scientific, pp. 153–167.

Borgatti, S. P. (2006), 'Identifying sets of key players in a social network', *Computational & Mathematical Organization Theory* **12**(1), 21–34.

Borgatti, S. P. & Everett, M. G. (2006), 'A graph-theoretic perspective on centrality', *Social networks* **28**(4), 466–484.

Brandes, U. (2008), 'On variants of shortest-path betweenness centrality and their generic computation', *Social Networks* **30**(2), 136–145.

Chemodanov, D., Esposito, F., Calyam, P. & Sukhov, A. (2018), 'A constrained shortest path scheme for virtual network service management', *IEEE Transactions on Network and Service Management* **16**(1), 127–142.

Chen, X. (2015), 'Critical nodes identification in complex systems', *Complex & Intelligent Systems* **1**(1-4), 37–56.

Consoli, S. & Darby-Dowman, K. (2006), Combinatorial optimization and metaheuristics, Technical report, Brunel University.

Corley, H. & David, Y. S. (1982), 'Most vital links and nodes in weighted networks', *Operations Research Letters* **1**(4), 157–160.

Cormican, K. J., Morton, D. P. & Wood, R. K. (1998), 'Stochastic network interdiction', *Operations Research* **46**(2), 184–197.

Cplex, I. I. (2009), 'V12. 1: User's manual for cplex', *International Business Machines Corporation* **46**(53), 157.

Crucitti, P., Latora, V., Marchiori, M. & Rapisarda, A. (2003), 'Efficiency of scale-free networks: error and attack tolerance', *Physica A: Statistical Mechanics and its Applications* **320**, 622–642.

Dantzig, G. B. (1951), 'Application of the simplex method to a transportation problem', *Activity analysis and production and allocation* .

Di Summa, M., Grosso, A. & Locatelli, M. (2011), 'Complexity of the critical node problem over trees', *Computers & Operations Research* **38**(12), 1766–1774.

Di Summa, M., Grosso, A. & Locatelli, M. (2012), 'Branch and cut algorithms for detecting critical nodes in undirected graphs', *Computational Optimization and Applications* **53**(3), 649–680.

Dinh, T. N. & Thai, M. T. (2015), 'Network under joint node and link attacks: Vulnerability assessment methods and analysis', *IEEE/ACM Transactions on Networking* **23**(3), 1001–1011.

Dinh, T. N., Xuan, Y., Thai, M. T., Pardalos, P. M. & Znati, T. (2012), 'On new approaches of assessing network vulnerability: hardness and approximation', *IEEE/ACM Transactions on Networking (ToN)* **20**(2), 609–619.

Du, W., Liang, B., Yan, G., Lordan, O. & Cao, X. (2017), 'Identifying vital edges in chinese air route network via memetic algorithm', *Chinese Journal of Aeronautics* **30**(1), 330–336.

Easley, D. & Kleinberg, J. (2010), *Networks, crowds, and markets: Reasoning about a highly connected world*, Cambridge University Press.

Elsayed, K. (2004), Hcasp: A hop-constrained adaptive shortest-path algorithm for routing bandwidth-guaranteed tunnels in mpls networks, *in* 'Proceedings. ISCC 2004. Ninth International Symposium on Computers And Communications (IEEE Cat. No. 04TH8769)', Vol. 2, IEEE, pp. 846–851.

Erdös, P. & Rényi, A. (1959), 'On random graphs', *Publicationes mathematicae* **6**(26), 290–297.

Ford Jr, L. R. & Fulkerson, D. R. (1962), *Flows in networks*, Princeton university press.

Girvan, M. & Newman, M. E. (2002), 'Community structure in social and biological networks', *Proceedings of the national academy of sciences* **99**(12), 7821–7826.

Gouveia, L., Joyce-Moniz, M. & Leitner, M. (2018), 'Branch-and-cut methods for the network design problem with vulnerability constraints', *Computers & Operations Research* **91**, 190–208.

Gouveia, L. & Leitner, M. (2017), 'Design of survivable networks with vulnerability constraints', *European Journal of Operational Research* **258**(1), 89–103.

Gouveia, L., Patrício, P. & de Sousa, A. (2006), Compact models for hop-constrained node survivable network design: An application to mpls, *in* 'Telecommunications planning: Innovations in pricing, network design and management', Springer, pp. 167–180.

Grötschel, M., Monma, C. L. & Stoer, M. (1995), 'Design of survivable networks', *Handbooks in operations research and management science* **7**, 617–672.

Guérin, R. & Orda, A. (2002), 'Computing shortest paths for any number of hops', *IEEE/ACM transactions on networking* **10**(5), 613–620.

Gurobi Optimization, I. (2018), 'Gurobi optimizer reference manual'.
URL: *http://www.gurobi.com*

Hagberg, A., Swart, P. & S Chult, D. (2008), Exploring network structure, dynamics, and function using networkx, Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States).

He, J., Liang, H. & Yuan, H. (2011), Controlling infection by blocking nodes and links simultaneously, *in* 'International workshop on internet and network economics', Springer, pp. 206–217.

Hernández-Pérez, H. & Salazar-González, J.-J. (2004), 'A branch-and-cut algorithm for a traveling salesman problem with pickup and delivery', *Discrete Applied Mathematics* **145**(1), 126–139.

Holme, P., Kim, B. J., Yoon, C. N. & Han, S. K. (2002), 'Attack vulnerability of complex networks', *Physical review E* **65**(5), 056109.

Hooshmand, F., Mirarabrazi, F. & MirHassani, S. (2020), 'Efficient benders decomposition for distance-based critical node detection problem', *Omega* **93**, 102037.

Hosoya, H. (1988), 'On some counting polynomials in chemistry', *Discrete applied mathematics* **19**(1-3), 239–257.

Hosteins, P. & Scatamacchia, R. (2020), 'The stochastic critical node problem over trees', *Networks* **76**(3), 321–426.

Huang, C.-Y. R., Lai, C.-Y. & Cheng, K.-T. T. (2009), Fundamentals of algorithms, *in* 'Electronic Design Automation', Elsevier, pp. 173–234.

Israeli, E. & Wood, R. K. (2002), 'Shortest-path network interdiction', *Networks* **40**(2), 97–111.

Kempe, D., Kleinberg, J. & Tardos, É. (2005), Influential nodes in a diffusion model for social networks, *in* 'International Colloquium on Automata, Languages, and Programming', Springer, pp. 1127–1138.

Khalil, E., Dai, H., Zhang, Y., Dilkina, B. & Song, L. (2017), Learning combinatorial otimization algorithms over graphs, *in* 'Advances in Neural Information Processing Systems', pp. 6348–6358.

Kitsak, M., Gallos, L. K., Havlin, S., Liljeros, F., Muchnik, L., Stanley, H. E. & Makse, H. A. (2010), 'Identification of influential spreaders in complex networks', *Nature physics* **6**(11), 888.

Knuth, D. E. (1993), *The Stanford GraphBase: a platform for combinatorial computing*, AcM Press New York.

Krishnamoorthy, M. S. & Deo, N. (1979), 'Node-deletion np-complete problems', *SIAM Journal on Computing* **8**(4), 619–625.

Lalou, M. & Kheddouci, H. (2019), 'A polynomial-time algorithm for finding critical nodes in bipartite permutation graphs', *Optimization Letters* **13**(6), 1345–1364.

Lalou, M., Tahraoui, M. A. & Kheddouci, H. (2016), 'Component-cardinality-constrained critical node problem in graphs', *Discrete Applied Mathematics* **210**, 150–163.

Lalou, M., Tahraoui, M. A. & Kheddouci, H. (2018), 'The critical node detection problem in networks: A survey', *Computer Science Review* **28**, 92–117.

Land, A. & Doig, A. (1960), 'An automatic method of solving discrete programming problems', *Econometrica* **28**(3), 497–520.

Lewis, J. M. & Yannakakis, M. (1980), 'The node-deletion problem for hereditary properties is np-complete', *Journal of Computer and System Sciences* **20**(2), 219–230.

Li, J., Pardalos, P. M., Xin, B. & Chen, J. (2019), 'The bi-objective critical node detection problem with minimum pairwise connectivity and cost: Theory and algorithms', *Soft Computing* **23**(23), 12729–12744.

Liu, H. & Wang, J. (2006), A new way to enumerate cycles in graph, *in* 'Advanced Int'l Conference on Telecommunications and Int'l Conference on Internet and Web Applications and Services (AICT-ICIW'06)', IEEE, pp. 57–57.

Ljubić, I., Weiskircher, R., Pferschy, U., Klau, G. W., Mutzel, P. & Fischetti, M. (2006), 'An algorithmic framework for the exact solution of the prize-collecting steiner tree problem', *Mathematical programming* **105**(2-3), 427–449.

Lusseau, D., Schneider, K., Boisseau, O. J., Haase, P., Slooten, E. & Dawson, S. M. (2003), 'The bottlenose dolphin community of doubtful sound features a large proportion of long-lasting associations', *Behavioral Ecology and Sociobiology* **54**(4), 396–405.

Mahjoub, A. R., Simonetti, L. & Uchoa, E. (2013), 'Hop-level flow formulation for the survivable network design with hop constraints problem', *Networks* **61**(2), 171–179.

Matisziw, T. C. & Murray, A. T. (2009), 'Modeling s–t path availability to support disaster vulnerability assessment of network infrastructure', *Computers & Operations Research* **36**(1), 16–26.

Mokken, R. J. et al. (1979), 'Cliques, clubs and clans', *Quality & Quantity* **13**(2), 161–173.

Morton, D. P. (2010), 'Stochastic network interdiction', *Wiley encyclopedia of operations research and management science* .

Murray, A. T., Matisziw, T. C. & Grubesic, T. H. (2007), 'Critical network infrastructure analysis: interdiction and system flow', *Journal of Geographical Systems* **9**(2), 103–117.

Myung, Y.-S. & Kim, H.-j. (2004), 'A cutting plane algorithm for computing k-edge survivability of a network', *European Journal of Operational Research* **156**(3), 579–589.

Nandi, A. K. & Medal, H. R. (2016), 'Methods for removing links in a network to minimize the spread of infections', *Computers & Operations Research* **69**, 10–24.

Naoum-Sawaya, J. & Buchheim, C. (2016), 'Robust critical node selection by benders decomposition', *INFORMS Journal on Computing* **28**(1), 162–174.

Neri, F. & Cotta, C. (2012), 'Memetic algorithms and memetic computing optimization: A literature review', *Swarm and Evolutionary Computation* **2**, 1–14.

Nguyen, D. T., Shen, Y. & Thai, M. T. (2013), 'Detecting critical nodes in interdependent power networks for vulnerability assessment', *IEEE Transactions on Smart Grid* **4**(1), 151–159.

Paton, M., Akartunalı, K. & Higham, D. J. (2017), 'Centrality analysis for modified lattices', *SIAM Journal on Matrix Analysis and Applications* **38**(3), 1055–1073.

Pereira, J., Ritt, M. & Vásquez, Ó. C. (2018), 'A memetic algorithm for the cost-oriented robotic assembly line balancing problem', *Computers & Operations Research* **99**, 249–261.

Pullan, W. (2015), 'Heuristic identification of critical nodes in sparse real-world graphs', *Journal of Heuristics* **21**(5), 577–598.

Purevsuren, D., Cui, G., Win, N. N. H. & Wang, X. (2016), 'Heuristic algorithm for identifying critical nodes in graphs', *Advances in Computer Science: an International Journal* **5**(3), 1–4.

Rocco, C. M. & Ramirez-Marquez, J. E. (2010), 'A bi-objective approach for shortest-path network interdiction', *Computers & Industrial Engineering* **59**(2), 232–240.

Rocco, C. M., Ramirez-Marquez, J. E. & Salazar, D. E. (2010), 'Bi and tri-objective optimization in the deterministic network interdiction problem', *Reliability Engineering & System Safety* **95**(8), 887–896.

Sadeghi, S., Seifi, A. & Azizi, E. (2017), 'Trilevel shortest path network interdiction with partial fortification', *Computers & Industrial Engineering* **106**, 400–411.

Salemi, H. & Buchanan, A. (2020), 'Solving the distance-based critical node problem', Optimization Online. `http://www.optimization-online.org/DB_HTML/2020/04/7751.html`.

Schieber, T. A., Carpi, L., Frery, A. C., Rosso, O. A., Pardalos, P. M. & Ravetti, M. G. (2016), 'Information theory perspective on network robustness', *Physics Letters A* **380**(3), 359–364.

Shahinpour, S. & Butenko, S. (2013), Distance-based clique relaxations in networks: s-clique and s-club, *in* 'Models, algorithms, and technologies for network analysis', Springer, pp. 149–174.

Shen, S. & Smith, J. C. (2012), 'Polynomial-time algorithms for solving a class of critical node problems on trees and series-parallel graphs', *Networks* **60**(2), 103–119.

Shen, S., Smith, J. C. & Goli, R. (2012), 'Exact interdiction models and algorithms for disconnecting networks via node deletions', *Discrete Optimization* **9**(3), 172–188.

Shen, Y., Nguyen, N. P., Xuan, Y. & Thai, M. T. (2013), 'On the discovery of critical links and nodes for assessing network vulnerability', *IEEE/ACM Transactions on Networking (TON)* **21**(3), 963–973.

Silver, E. A. (2004), 'An overview of heuristic solution methods', *Journal of the operational research society* **55**(9), 936–956.

Smith, J. C. & Song, Y. (2020), 'A survey of network interdiction models and algorithms', *European Journal of Operational Research* **283**(3), 797–811.

Tomaino, V., Arulselvan, A., Veltri, P. & Pardalos, P. M. (2012), Studying connectivity properties in human protein–protein interaction network in cancer pathway, *in* 'Data Mining for Biomarker Discovery', Springer, pp. 187–197.

UCINET software datasets (n.d.), `https://sites.google.com/site/ucinetsoftware/datasets/`. Last accessed: 06-11-2018.

Ventresca, M. (2012), 'Global search algorithms using a combinatorial unranking-based problem representation for the critical node detection problem', *Computers & Operations Research* **39**(11), 2763–2775.

Ventresca, M. & Aleman, D. (2014), A fast greedy algorithm for the critical node detection problem, *in* 'International Conference on Combinatorial Optimization and Applications', Springer, pp. 603–612.

Ventresca, M. & Aleman, D. (2015), 'Efficiently identifying critical nodes in large complex networks', *Computational Social Networks* **2**(1), 6.

Ventresca, M., Harrison, K. R. & Ombuki-Berman, B. M. (2018), 'The bi-objective critical node detection problem', *European Journal of Operational Research* **265**(3), 895–908.

Veremyev, A., Boginski, V. & Pasiliao, E. L. (2014), 'Exact identification of critical nodes in sparse networks via new compact formulations', *Optimization Letters* **8**(4), 1245–1259.

Veremyev, A., Prokopyev, O. A. & Pasiliao, E. L. (2014), 'An integer programming framework for critical elements detection in graphs', *Journal of Combinatorial Optimization* **28**(1), 233–273.

Veremyev, A., Prokopyev, O. A. & Pasiliao, E. L. (2015), 'Critical nodes for distance-based connectivity and related problems in graphs', *Networks* **66**(3), 170–195.

Walteros, J. L., Veremyev, A., Pardalos, P. M. & Pasiliao, E. L. (2019), 'Detecting critical node structures on graphs: A mathematical programming approach', *Networks* **73**(1), 48–88.

Wang, S., Gong, M., Liu, W. & Wu, Y. (2020), 'Preventing epidemic spreading in networks by community detection and memetic algorithm', *Applied Soft Computing* **89**, 106–118.

Wasserman, S. & Faust, K. (1994), *Social network analysis: Methods and applications*, Vol. 8, Cambridge university press.

Williams, H. P. (1986), 'Fourier's method of linear programming and its dual', *The American mathematical monthly* **93**(9), 681–695.

Wolsey, L. A. (1998), *Integer programming*, John Wiley & Sons.

Wolsey, L. A. & Nemhauser, G. L. (1999), *Integer and combinatorial optimization*, Vol. 55, John Wiley & Sons.

Wood, R. K. (1993), 'Deterministic network interdiction', *Mathematical and Computer Modelling* **17**(2), 1–18.

Yadegari, E., Alem-Tabriz, A. & Zandieh, M. (2019), 'A memetic algorithm with a novel neighborhood search and modified solution representation for closed-loop supply chain network design', *Computers & Industrial Engineering* **128**, 418–436.

Yannakakis, M. (1978), Node-and edge-deletion np-complete problems, *in* 'Proceedings of the tenth annual ACM symposium on Theory of computing', pp. 253–264.

Zachary, W. W. (1977), 'An information flow model for conflict and fission in small groups', *Journal of anthropological research* **33**(4), 452–473.

Zhang, L., Xia, J., Cheng, F., Qiu, J. & Zhang, X. (2020), 'Multi-objective optimization of critical node detection based on cascade model in complex networks', *IEEE Transactions on Network Science and Engineering* .

Zhou, B., Cai, X. & Trinajstić, N. (2008), 'On harary index', *Journal of mathematical chemistry* **44**(2), 611–618.

Zhou, Y., Hao, J.-K. & Glover, F. (2018), 'Memetic search for identifying critical nodes in sparse graphs', *IEEE transactions on cybernetics* **49**(10), 3699–3712.

Appendix A

Codes

A.1 Aggregated Path-based model (DCNDP-PBM)

```
import networkx as nx
from gurobipy.gurobipy import Model, GRB, LinExpr
import dist_connectivity as dc
#-------Lazy cut callback to separate path constraints--------------------
def mycut(model, where):
    global savebnd, cutcount, bndcheck
    cost = {}
    connect={}

    #-----separates integer solution
    def depthKsp1(graph, cost, connect, L, cutlimit):
        roots=[n for (n,attr) in cost.items() if attr < 1-1e-5]
        input_graph=graph.subgraph(roots)
        for rt in roots:
            cutcount=0
            length, path = nx.single_source_dijkstra(input_graph,rt,
                                                     cutoff=L,weight='weight')
            for v,distance in length.items():
                if rt!=v:
                    i=min([rt,v])
                    j=max([rt,v])
```

```
                    if connect[(i, j)] < f[distance-1]:
                        model.cbLazy((f[distance-1]*\
                        sum(model._x_delete[node] for node in path[v])) + \
                model._u_connect[i,j] >= f[distance-1])
                        cutcount+=1
                        if cutcount ==cutlimit:
                            break
    #-----separates fractional solution
    def depthKsp2(graph, cost, connect, L, cutlimit):
        roots=[n for (n,attr) in cost.items() if attr < 1-1e-5]
        input_graph=graph.subgraph(roots)
        for rt in roots:
            cutcount=0
            length, path = nx.single_source_dijkstra(input_graph,rt,
                                                     cutoff=L,weight='weight')
            for v,distance in length.items():
                if rt!=v:
                    i=min([rt,v])
                    j=max([rt,v])
                    if (f[distance-1]*sum(cost[node] for node in path[v])) +\
                        connect[(i, j)] < f[distance-1]:
                        model.cbLazy((f[distance-1]*\
                        sum(model._x_delete[node] for node in path[v])) + \
                        model._u_connect[i,j] >= f[distance-1])
                        cutcount+=1
                        if cutcount ==cutlimit:
                            break
    #if integer solution
    if where == GRB.Callback.MIPSOL:
         savebnd=model.cbGet(GRB.callback.MIPSOL_OBJBND)
         nodecnt = model.cbGet(GRB.Callback.MIPSOL_NODCNT)
         for j in G.nodes():
            cost[j]=abs(model.cbGetSolution(model._x_delete[j]))
            for i in G.nodes():
                if i<j:
                    connect[(i,j)] = \
```

```
                    abs(model.cbGetSolution(model._u_connect[i,j]))
        depthKsp1(G, cost, connect, L, GRB.INFINITY)

    # if fractional solution
    elif where ==GRB.Callback.MIPNODE:
        if model.cbGet(GRB.Callback.MIPNODE_STATUS) == GRB.Status.OPTIMAL:
            currentbnd=model.cbGet(GRB.callback.MIPNODE_OBJBND)
            nodecnt=int(model.cbGet(GRB.callback.MIPNODE_NODCNT))
            if savebnd==currentbnd:
                bndcheck+=1
                if bndcheck >=5 or nodecnt>0:
                    bndcheck=0
                else:
                    for j in G.nodes():
                        cost[j]=abs(model.cbGetNodeRel(model._x_delete[j]))
                        for i in G.nodes():
                            if i<j:
                                connect[(i,j)] = \
                                abs(model.cbGetNodeRel(model._u_connect[i,j]))
                    depthKsp2(G, cost, connect, L, 300)
            else:
                savebnd=currentbnd
                for j in G.nodes():
                        cost[j]=abs(model.cbGetNodeRel(model._x_delete[j]))
                        for i in G.nodes():
                            if i<j:
                                connect[(i,j)] = \
                                abs(model.cbGetNodeRel(model._u_connect[i,j]))
                depthKsp2(G, cost, connect, L, 300)

#-----Minimize distance-based connectivity metric specified by f((l)----------
def main(H,k,C):
    model = Model('Minimize distance-based connectivity objective')

   #variables
    x_delete = {}
```

```
u_connect = {}
for j in H.nodes():
    if H.degree[j]==1:
        x_delete[j] = model.addVar(lb=0.0, ub=0.0, vtype=GRB.BINARY,
                name ="x[%d]" %(j))
    else:
        x_delete[j] = model.addVar(lb=0.0, ub=1.0, vtype=GRB.BINARY,
                name ="x[%d]" %(j))

    for i in H.nodes():
        if i <j:
            u_connect[i,j] =model.addVar(lb=0.0, ub=1.0,
                    vtype=GRB.CONTINUOUS, name="u[%d,%d]" %(i,j))

#objective function
obj = LinExpr(0)
for j in H.nodes():
    for i in H.nodes():
        if i<j:
            obj.add(u_connect[i,j])

#constraint on number of critical nodes
model.addConstr(sum((x_delete[j]) for j in H.nodes())<=C)

#constraints on connectivity variables u
#constraints on (i,j) in E
for (i,j) in H.edges():
     weight=H.edges[i, j]['weight']
     if i<j:
        model.addConstr(u_connect[i,j] + f[weight-1]*(x_delete[i]+ \
                        x_delete[j]) >= f[weight-1])
     else: #that is j<i
         model.addConstr(u_connect[j,i] + f[weight-1]*(x_delete[j]+ \
                         x_delete[i]) >= f[weight-1])
model.update()
model.setObjective(obj, GRB.MINIMIZE)
```

```
    model._x_delete=x_delete
    model._u_connect=u_connect
    model.setParam(GRB.param.Cuts, 0)
    model.setParam('LogToConsole', 0)
    model.setParam(GRB.param.PreCrush, 1)
    model.setParam('LazyConstraints', 1)
    model.setParam('TimeLimit', 3600)
    #model.write("DCNP-gen.lp")
    model.optimize(mycut)
    run_time=model.Runtime
    xval=model.getAttr('x', x_delete)

    critical_nodes=[i for i in xval.keys() if xval[i]>=1- 1e-4]

    opt_obj = 0
    for j in H.nodes():
        for i in H.nodes():
            if i < j:
                opt_obj+= u_connect[i, j].X

    return critical_nodes,opt_obj, run_time, model.Runtime

#------------------Main body------------------------------
G=nx.read_edgelist(path="SmallWorld.edgelist", nodetype=int)

#assign weights to edges based on edge betweenness
edge_btw=nx.edge_betweenness_centrality(G, normalized=True)
for e in G.edges():
    G.edges[e[0],e[1]]['weight'] = min(2,max(1,round(0.1/edge_btw[e])))

bndcheck=0
L=dc.cal_diameter(G) #diameter of the graph
n=G.number_of_nodes()
C=int(0.05*n) #budget on critical nodes
ind=0
```

```
#define distance connectivity function (eg f(d)=1/d)
f=[]
for l in range(L+1):
    f.append(1/float(l+1))

#find the critical nodes
critical_nodes,opt_obj,run_time,cpu_time=main(G,L,C)
print("critical nodes are :", critical_nodes)
print("Running Time = {:.2f} seconds".format(run_time))
print('Final objective = {:.2f}'.format(opt_obj))
print('Final objective percentage = {:.2f}%'.format(2*100*opt_obj/(n*(n-1))))
```

A.2 Disaggregated Path-based model (DCNDP-PBML)

```
import networkx as nx
from gurobipy.gurobipy import Model, GRB, LinExpr
import dist_connectivity as dc
#-------Lazy cut callback to separate path constraints--------------------
def mycut(model, where):
    global savebnd, cutcount, bndcheck
    cost = {}
    connect={}
    #-----separates integer solution -----
    def depthKsp1(graph, cost, connect, L, cutlimit):
        roots=[n for (n,attr) in cost.items() if attr < 1-1e-5]
        input_graph=graph.subgraph(roots)
        for rt in roots:
            cutcount=0
            length, path = nx.single_source_dijkstra(input_graph, rt,
                                                     cutoff=L, weight='weight')
            for v,distance in length.items():
                if rt!=v:
                    i=min([rt,v])
                    j=max([rt,v])
                    if connect[(i, j, distance)] < 1:
```

```
                    model.cbLazy(sum(model._x_delete[n] for n in path[v])\
                    + model._u_connect[i,j,distance] >= 1)
                    cutcount+=1
                    if cutcount ==cutlimit:
                        break

#separates fractional solution
def depthKsp2(graph, cost, connect, L, cutlimit):
    roots=[n for (n,attr) in cost.items() if attr < 1-1e-5]
    input_graph=graph.subgraph(roots)
    for rt in roots:
        cutcount=0
        length, path = nx.single_source_dijkstra(input_graph, rt,
                                                 cutoff=L, weight='weight')
        for v,distance in length.items():
            if rt!=v:
                i=min([rt,v])
                j=max([rt,v])
                if sum(cost[n] for n in path[v]) + \
                connect[(i, j, distance)] < 1:
                    model.cbLazy(sum(model._x_delete[n] for n in path[v])\
                    +  model._u_connect[i,j,distance] >= 1)
                    cutcount+=1
                    if cutcount ==cutlimit:
                        break

#if integer solution
if where == GRB.Callback.MIPSOL:
     savebnd=model.cbGet(GRB.callback.MIPSOL_OBJBND)
     nodecnt = model.cbGet(GRB.Callback.MIPSOL_NODCNT)
     for j in G.nodes():
        cost[j]=abs(model.cbGetSolution(model._x_delete[j]))
        for i in G.nodes():
                for l in range(1,L+1):
                    connect[(i,j,l)] = \
                    abs(model.cbGetSolution(model._u_connect[i,j,l]))
```

```
            depthKsp1(G, cost, connect, L, GRB.INFINITY)

    #if fractional solution
    elif where ==GRB.Callback.MIPNODE:
            if model.cbGet(GRB.Callback.MIPNODE_STATUS) == GRB.Status.OPTIMAL:
               currentbnd=model.cbGet(GRB.callback.MIPNODE_OBJBND)
               nodecnt=int(model.cbGet(GRB.callback.MIPNODE_NODCNT))
               if savebnd==currentbnd:
                   bndcheck+=1
                   if bndcheck >=5 or nodecnt>0:
                      bndcheck=0
                   else:
                       for j in G.nodes():
                           cost[j]=abs(model.cbGetNodeRel(model._x_delete[j]))
                           for i in G.nodes():
                               if i<j:
                                   for l in range(1,L+1):
                                       connect[(i,j,l)] = \
                                       abs(model.cbGetNodeRel(model._u_connect[i,j,l]))
                       depthKsp2(G, cost, connect, L, 300)

               else:
                   savebnd=currentbnd
                   for j in G.nodes():
                           cost[j]=abs(model.cbGetNodeRel(model._x_delete[j]))
                           for i in G.nodes():
                               if i<j:
                                   for l in range(1,L+1):
                                       connect[(i,j,l)] = \
                                       abs(model.cbGetNodeRel(model._u_connect[i,j,l]))
                   depthKsp2(G, cost, connect, L, 300)


#-----Minimize distance-based connectivity metric specified by f((l)---------
def main(H,k,C):
    model = Model('Minimize distance-based connectivity objective')
```

```
#Decision variables
 x_delete = {}
 u_connect = {}
 for j in H.nodes():
     if H.degree[j]==1:
         x_delete[j] = model.addVar(lb=0.0, ub=0.0,
                 vtype=GRB.BINARY, name ="x[%d]" %(j))
     else:
         x_delete[j] = model.addVar(lb=0.0, ub=1.0,
                 vtype=GRB.BINARY, name ="x[%d]" %(j))

     for i in H.nodes():
         if i <j:
             for l in range(1,L+1):
                 u_connect[i, j, l] =model.addVar(lb=0.0, ub=1.0,
                          vtype=GRB.BINARY, name="u[%d,%d,%d]" %(i,j,l))

 #objective function
 obj = LinExpr(0)
 for j in H.nodes():
     for i in H.nodes():
         if i<j:
             obj.add(f[0] * u_connect[i, j, 1])
             for l in range(1,L):
                 obj.add(f[l] * (u_connect[i, j, l + 1] - u_connect[i, j, l]))

 #constraint on number of critical nodes
 model.addConstr(sum((x_delete[j]) for j in H.nodes())<=C)


 #constraints on (i,j) in E
 for (i,j) in H.edges():
      weight_ij=H.edges[i, j]['weight']
      if i<j:
         model.addConstr(u_connect[i,j,weight_ij] + x_delete[i]+ \
```

```
                            x_delete[j] >= 1)
            weights_other=[H.edges[i,t]['weight'] for t in set(H[i])-set([j])]
            weights_other.append(weight_ij-1)
            weight=min(weights_other)
            for l in range(1, weight+1):
                model.addConstr(u_connect[i,j,l] == 0)
            for l in range(weight_ij+1, L+1):
                model.addConstr(u_connect[i,j,l] == u_connect[i,j,weight_ij])
        else: #that is j<i
            model.addConstr(u_connect[j,i,weight_ij] + \
                            x_delete[j]+ x_delete[i] >= 1)
            weights_other=[H.edges[j,t]['weight'] for t in set(H[j])-set([i])]
            weights_other.append(weight_ij-1)
            weight=min(weights_other)
            for l in range(1, weight+1):
                model.addConstr(u_connect[j,i,l] == 0)
            for l in range(weight_ij+1, L+1):
                model.addConstr(u_connect[j,i,l] == u_connect[j,i,weight_ij])

#constraints on (i,j) not in E
for j in H.nodes():
    for i in H.nodes():
        if i not in H.neighbors(j) and i<j:
            for l in range(2, L):
                model.addConstr(u_connect[i,j,l] <= u_connect[i,j,l+1])


model.update()
model.setObjective(obj, GRB.MINIMIZE)
model._x_delete=x_delete
model._u_connect=u_connect
model.setParam(GRB.param.Cuts, 0)
model.setParam('LogToConsole', 0)
model.setParam(GRB.param.PreCrush, 1)
model.setParam('LazyConstraints', 1)
model.setParam('TimeLimit', 3600)
```

```
    model.optimize(mycut)
    run_time=model.Runtime
    xval=model.getAttr('x', x_delete)

    critical_nodes=[i for i in xval.keys() if xval[i]>=1- 1e-4]

    opt_obj = 0
    for j in H.nodes():
        for i in H.nodes():
            if i < j:
                opt_obj+=f[0] * u_connect[i, j, 1].X
                for l in range(1,L):
                    opt_obj+=f[l] * (u_connect[i, j, l + 1].X - \
                             u_connect[i, j, l].X)

    return critical_nodes,opt_obj, run_time, model.Runtime

#------------------Main body------------------------------
G=nx.read_edgelist(path="USAir97.edgelist", nodetype=int)
#assign weights to edges based on edge betweenness------------
edge_btw=nx.edge_betweenness_centrality(G, normalized=True)
for e in G.edges():
    G.edges[e[0],e[1]]['weight'] = min(2,max(1,round(0.1/edge_btw[e])))

bndcheck=0

n=G.number_of_nodes()
L=dc.cal_diameter(G)
C=int(0.05*n)
ind=0

#define distance connectivity function (eg f(d)=1/d)
f=[]
for l in range(L+1):
    f.append(1/float(l+1))
```

```
#find the critical nodes
critical_nodes,opt_obj,run_time,cpu_time=main(G,L,C)
print("critical nodes are :", critical_nodes)
print("Running Time = {:.2f} seconds".format(run_time))
print('Final objective = {:.2f}'.format(opt_obj))
print('Final objective percentage = {:.2f} %'.format(2*100*opt_obj/(n*(n-1))))
```

A.3 Heuristic Algorithm

```
from functools import wraps
from time import time
import networkx as nx
import collections
import dist_connectivity as ds
from operator import itemgetter
import random

def timing(f):
    @wraps(f)
    def wrapper(*args, **kwargs):
        global runTime
        start = time()
        result = f(*args, **kwargs)
        end = time()
        runTime=round(end-start,1)
        return result
    return wrapper

def k_distBFS(input_graph, root, k):
    #keep track of all visited nodes and nodes to be checked
    visited, queue =set([root]), collections.deque([root])
    levels = {}          # this dict keeps track of levels
    levels[root]= 0     # depth of root node is 0

    pred={}  #this dict keeps track of predecessors
```

```
    pred[root]=-1 #predecessor of root node is -1
    while queue: #keep looping until there are no nodes still to be checked
        vertex = queue.popleft() #pop first node from the queue
        for neighbour in input_graph[vertex]:
            if neighbour not in visited:
                newlevel=levels[vertex]+1
                if newlevel>k:
                    break
                else:
                    pred[neighbour]=vertex
                    levels[neighbour]=newlevel
                    #mark neighbours of node as visited to avoid revisiting
                    visited.add(neighbour)
                    queue.append(neighbour) #add neighbours of node to queue
        else:
            continue
        break
    del pred[root]

    return list(pred.values())

#--Randomising node selection from node sets ranked according to their centrality
def randomize_centrality(centrality, randomSize):
    randFlag=random.choices([1,2], weights=[p1,p2], k=randomSize)
    nodecnt=0
    centralNodes=[]
    centralNbr=[]
    for flag in randFlag:
        if flag==1:
            centralNodes.append(centrality[nodecnt])
        else:
            centralNbr.append(centrality[nodecnt])
        nodecnt+=1

    centralNodes.extend(centralNbr)
    return centralNodes
```

```
#--Greedy approach to repair offspring solution
def greedy_BBcrossover(minconn,critNodes,nbr):
    for i in nbr:
         subGnodes=[n for n in G.nodes() if (n not in critNodes and n!=i)]
         kconn=ds.subG_kconnectivity(G,k,subGnodes)
         if kconn < minconn:
            addNode=i
            minconn=kconn
    return addNode,minconn

#--Generate initial solution based on centrality and do backbone crossover
def centralityHeur(G,k,C):
    global currentObj
    global solA1
    kBFS_size={}
    degree_centrality = [n for n, d in sorted(G.degree(),\
                                    key=itemgetter(1), reverse=True)]
    root=degree_centrality[0]
    tree_nodes=k_distBFS(G, root, k)
    kBFS_size[root]=len(tree_nodes)
    kbetweeenness_Count=collections.Counter(tree_nodes)
    for rt in degree_centrality[1:n]:
       tree_nodes=k_distBFS(G, rt, k)
       kBFS_size[rt]=len(tree_nodes)
       node_kbetweenness=collections.Counter(tree_nodes)
       kbetweeenness_Count.update(node_kbetweenness)
    kBFS=collections.Counter(kBFS_size)

    kbtw_centrality=[n for n,d in kbetweeenness_Count.most_common()]
    kkatz_centrality=[n for n,d in kBFS.most_common()]

    #gnerate initial solutions based on centrality measures
    solA=randomize_centrality(degree_centrality, B)
    solB=randomize_centrality(kkatz_centrality, B)
    solC=randomize_centrality(kbtw_centrality, B)
```

```
solA1=solA[:C] #heuristic solution based on degree centrality
solB1=solB[:C] #heuristic solution based on k-katz centrality
solC1=solC[:C] #heuristic solution based on k-betweenness centrality

unionExtSol=set(solA).union(set(solB),set(solC))
unionSol=set(solA1).union(set(solB1),set(solC1))
two_par1=set(solA1).intersection(set(solB1))
two_par2=set(solA1).intersection(set(solC1))
two_par3=set(solC1).intersection(set(solB1))
union_two_parent=two_par1.union(two_par2,two_par3)

three_parent=two_par1.intersection(two_par2,two_par3)# 3-parent nodes

#return the offspring solution if the size is equal to budget
if len(three_parent) == C:
    subGnodes=[n for n in G.nodes() if n not in list(three_parent)]
    currentObj=ds.subG_kconnectivity(G,k,subGnodes)
    return currentObj,three_parent

#else repair offspring solution via backbone crossover
else:
    two_parent=union_two_parent.difference(three_parent)#2-parent nodes
    one_parent=unionSol.difference(union_two_parent)# 1-parent nodes
    no_parent=unionExtSol.difference(unionSol)#0-parent nodes

    offspring=three_parent.copy()
    offspring_nbr=two_parent.union(one_parent,no_parent)

    subGnodes=[n for n in G.nodes() if n not in list(offspring)]
    currentObj=ds.subG_kconnectivity(G,k,subGnodes)

    randCrossover=random.choices([1,2], weights=[0.7,0.3], k=C-len(offspring))
    for rand1 in randCrossover:

        if rand1==1:#use greedy crossover
```

```
                includeNode, Obj=greedy_BBcrossover(currentObj,
                                                    offspring,list(offspring_nbr))
                offspring.add(includeNode)
                offspring_nbr.remove(includeNode)
                currentObj=Obj

            else:#randomly select from 2-parent, 1-parent, 0-parent nodes
                rand2=random.choices([1,2,3], weights=[0.5,0.3,0.2], k=1)

                #select from 2-parent
                if rand2==1 and len(two_parent.difference(offspring))!=0:
                    includeNode=random.choice(list(two_parent.difference(offspring)))
                    offspring.add(includeNode)
                    offspring_nbr.remove(includeNode)

                #select from 1-parent
                elif rand2==2 and len(one_parent.difference(offspring))!=0:
                     includeNode=random.choice(list(one_parent.difference(offspring)))
                     offspring.add(includeNode)
                     offspring_nbr.remove(includeNode)

                #select from 0-parent
                elif rand2==3 and len(no_parent.difference(offspring))!=0:
                     includeNode=random.choice(list(no_parent.difference(offspring)))
                     offspring.add(includeNode)
                     offspring_nbr.remove(includeNode)

                else:
                    includeNode=random.choice(list(offspring_nbr.difference(offspring)))
                    offspring.add(includeNode)
                    offspring_nbr.remove(includeNode)
                subGnodes=[n for n in G.nodes() if n not in list(offspring)]
                currentObj=ds.subG_kconnectivity(G,k,subGnodes)
        return currentObj,offspring

#-------------------------------------------------------------
```

```
def centrality_nbrhood(H):
    kBFS_size={}
    degree_centrality = [n for n, d in sorted(H.degree(),
                                              key=itemgetter(1), reverse=True)]
    root=degree_centrality[0]
    tree_nodes=k_distBFS(H, root, k)
    kBFS_size[root]=len(tree_nodes)
    kbetweeenness_Count=collections.Counter(tree_nodes)
    for rt in degree_centrality[1:int(0.5*n)]:
       tree_nodes=k_distBFS(H, rt, k)
       kBFS_size[rt]=len(tree_nodes)
       node_kbetweenness=collections.Counter(tree_nodes)
       kbetweeenness_Count.update(node_kbetweenness)
    kBFS=collections.Counter(kBFS_size)

    kbtw_centrality=[n for n,d in kbetweeenness_Count.most_common()]
    kkatz_centrality=[n for n,d in kBFS.most_common()]

    # degree centrality neighbourhood
    nbr1=randomize_centrality(degree_centrality, B)

    # k-betweenness centrality neighbourhood
    nbr2=randomize_centrality(kkatz_centrality, B)

    # k-Katz centrality neighbourhood
    nbr3=randomize_centrality(kbtw_centrality, B)

    unionNbr=set(nbr1).union(set(nbr2),set(nbr3))
    return list(unionNbr)

#---identify node to swap with a node in the current feasible solution---------
def swap(minObj,n,swapNodes):
    for i in swapNodes:
        otherNodes=[node for node in G.nodes() if (node not in swapNodes or node==i)]
        newObj=ds.subG_kconnectivity(G,k,otherNodes)
        if newObj <= minObj:
```

```
            removeNode=i
            minObj=newObj

    return removeNode,minObj
#----centrality-based neighbourhood search using two-phase node swap----
def nbr_search(initObj,initSol):
    global bestObj
    global iterCnt
    subGnodes=[n for n in G.nodes() if n not in initSol]
    bestObj=initObj
    bestSol=initSol.copy()
    subG=G.subgraph(subGnodes)
    nbrhood=centrality_nbrhood(subG)
    nbrCnt=0
    iterCnt=0
    while nbrCnt < len(nbrhood) and iterCnt < maxIter:
        nbr=nbrhood[nbrCnt]
        initSol.add(nbr)
        swapNode, newObj=swap(bestObj,nbr,list(initSol))
        nbrCnt+=1
        initSol.remove(swapNode)
        if newObj < bestObj:
            bestObj=newObj
            bestSol=initSol.copy()
            iterCnt=0
        else:
            iterCnt+=1
    return bestObj,bestSol
#-------------------------------------------
@timing
def completeHeur(G,k,C):
    global best_obj
    CurrBestObj,CurrBestSol=centralityHeur(G,k,C)
    best_obj,bestSol=nbr_search(CurrBestObj,CurrBestSol)
    return best_obj
```

```
G=nx.read_edgelist(path="USAir97.edgelist", nodetype=int)
n=G.number_of_nodes()
C=int(0.1*n)
p1=0.9
p2=0.1
k=3
eps=max(5,int(0.2*C))
B=C+eps
maxIter=100
print("finalSol, initialSol and runtime: ", \
      completeHeur(G,k,C), currentObj," and ", runTime)
```