Università degli Studi di Padova
Department of Mathematics "Tullio Levi-Civita"
Doctoral Program in Mathematical Sciences
Curriculum: Mathematics
Cycle XXXIII

Ph.D. Thesis

Variants of the Critical Node/Edge Detection Problem on a tree

---

PhD Coordinator: Prof. Martino Bardi

Supervisor: Prof. Marco Di Summa

Candidate: Syed Md Omar Faruk

September, 2021

To my parents

Acknowledgements

All praise and gratitude to Allah, who has provided me with the energy and ability to finish my task effectively.

Throughout my PhD period, I am grateful to my supervisor, Professor Marco Di Summa, for his constant right and excellent direction, inspiration, caring support, and helpful recommendations. I feel extremely fortunate to have the opportunity to learn from and be supported by such a mentor. He helps me become a better mathematician and guides me in the proper route.

I would want to express my gratitude to Prof. Martino Bardi for his assistance from the outset when I was having difficulty obtaining a visa and making critical decisions at various points throughout these three years. Prof. Bardi also helped me choose my advisor by pointing me in the appropriate way.

My sincere thanks to Mrs. Loretta Dalla Costa, the secretary of the Ph.D. course of the Mathematics Department of the University of Padova for her valuable assistance in the various administrative issues. Gratitude is due to the office staff of the Department of Mathematics, UNIPD, for their all-out support and services.

I would also like to express my gratitude to all of the PhD students and researchers I met throughout my time at UNIPD's Mathematics department. Furthermore, I extend my heartfelt gratitude to all of the people who have surrounded me during my time in Padova for their unwavering support, inspiration, and assistance in focusing spontaneously to complete my thesis work.

Finally, my heartfelt thanks go to my loving parents, two younger sisters, and lovely wife for their love, prayers, and spiritual support in allowing me to pursue my aspirations. I would want to express my gratitude to my wife Kamrun Nahar Jabin, who has cared for our kid (Syed Usman Bin Faruk) alone since he was just 80 days old when I arrived in Italy to begin my PhD. There are no words to adequately express my gratitude to my family for all of their efforts on my behalf.

Abstract

We consider the problem of removing a limited subset of nodes and/or edges from a graph G in order to minimize the so-called pairwise connectivity of the residual graph, which is defined as the total cost of the pairs of nodes still connected by a path. This is a well-studied version of a family of problems known as critical node or edge detection problems. The main inspiration and common interest behind these topics stems from their relevance in numerous domains, including computational biology, transportation problems, the assessment of network security, and many others.

In this thesis we study the case in which the graph G has a hierarchical organization, so that G is a tree. We begin by considering some natural variants of the critical node detection problem for special graph class such as paths and present an exact approach to solve the problems. We continue by investigating the critical edge detection problem over trees in different situations of edge weights. We provide polynomial algorithms for the problems when all connections between pairs of nodes have unit cost.

Indeed, while most of the literature focuses on deleting nodes or edges separately, we allow the simultaneous removal of nodes and edges. We consider both the case in which the nodes and edges removed must satisfy a joint weight limit, and the case in which two separate weight limits are given for nodes and edges. We also consider our problems to a more general setting, in which the connection costs are called "square 0/1 connection costs". We then explore our problems with arbitrary 0/1 connection costs and nonnegative node/edge weights when the number of leaves in the tree is fixed. We study the complexity of several problems of this type when the given graph is a tree, providing NP-hardness results or polynomial-time algorithms for the different cases that we analyze. In addition, we present mathematical models based on integer linear programming which provides optimal solutions for critical node and/or edge problems.

Finally we consider a generalization of the pairwise connectivity critical node/edge detection problem, the so-called Distance-based Critical Node/Edge Detection Problem. We investigate certain versions of our problems that fall into this category and propose polynomial time algorithms. We also develop a mathematical framework for the distance-based critical edge identification problem based on integer linear programming.

Keywords: Critical node detection, Critical edge detection, Combinatorial optimization, Complexity, Dynamic programming, Integer linear programming, Network interdiction, Pairwise connectivity.

Sommario (Italian abstract)

In questo lavoro consideriamo il problema di rimuovere un sottoinsieme limitato di nodi e/o archi da un grafo G così da minimizzare una misura della connettività del grafo residuo, definita come il costo totale delle coppie di nodi ancora connesse da un cammino. Questa è una versione molto studiata di una famiglia di problemi noti come problemi di identificazione di nodi o archi critici. La principale motivazione e l'interesse comune alla base di questi argomenti nascono dalla loro rilevanza in numerosi campi, tra cui la biologia computazionale, problemi di trasporto, la valutazione della sicurezza delle reti e molti altri.

In questa tesi studiamo il caso in cui il grafo G ha una struttura gerarchica, per cui G è un albero. Cominceremo considerando alcune varianti naturali del problema di identificazione di nodi critici per alcune classi particolari di grafi, come i cammini, e presentiamo un approccio esatto per risolvere questi problemi. Continueremo studiando il problema di identificazione di archi critici su alberi con diverse tipologie di pesi sugli archi. Otterremo algoritmi polinomiali per questi problemi quando tutte le connessioni tra coppie di nodi hanno costo unitario.

Di fatto, mentre gran parte della letteratura è incentrata sulla rimozione di nodi o archi separatamente, qui consentiamo la rimozione contemporanea di nodi e archi. Considereremo sia il caso in cui i nodi e gli archi rimossi devono soddisfare un singolo vincolo cumulativo sul loro peso, sia il caso in cui vengono assegnati due vincoli separati per il peso di nodi e archi rimossi. Studieremo i nostri problemi anche in un contesto più generale, in cui le connessioni hanno una struttura che chiameremo "costi 0/1 quadrati". Esploreremo poi questi problemi con costi di connessione 0/1 generali e pesi non-negativi su nodi e/o archi sotto l'ipotesi che il numero di foglie dell'albero sia fissato. Studieremo anche la complessità di vari problemi di questo tipo quando il grafo è un albero, fornendo risultati di NP-completezza o algoritmi polinomiali a seconda del caso in esame. Inoltre, presenteremo modelli matematici basati sulla programmazione lineare intera che forniscono soluzioni ottime per problemi di rimozione di nodi e/o archi.

Infine considereremo una generalizzazione della misura della connettività considerata sopra, che dà luogo a problemi di identificazione di nodi e archi critici basati sulla distanza. Studieremo certe versioni dei nostri problemi che cadono in questa categoria e proporremo algoritmi polinomiali. Svilupperemo anche una formulazione matematica per questi problemi tramite programmazione lineare intera.

Parole chiave: Identificazione di nodi critici, Identificazione di archi critici, Ottimizzazione combinatoria, Complessità, Programmazione dinamica, Programmazione lineare intera, Network interdiction, Connettività a coppie.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Critical node or edge detection problems are a family of optimization problems defined on graphs, where one is required to remove a limited number of nodes and/or edges in order to minimize some measure of the connectivity of the residual graph. This class of problems has attracted the interest of many researchers in the last two decades, because of its relevance in a number of practical applications. The applications involve situations where the aim is to either protect the connectivity of nodes in a network by securing the most critical nodes or attacking the most critical nodes in order to have minimum connections between all pairs of nodes in the network. Some important applications of the critical node or edge detection problems are presented below.

In a supply chain network, the connections between pairs of nodes is minimized after removing the most critical nodes from the network. For example, a military supply chain network [88] contains battalions and support battalions as nodes and the connections between them as links. By attacking the most critical nodes in this network, the connectivity between supply and demand nodes will be minimized. Therefore, the solution of the critical node detection problem (CNDP) is important in military tactical attacks during wars.

By using the gathered intelligence from a covert network, the terrorist network can be represented as a graph where terrorists are depicted as nodes and the social interactions between them represent links. We can minimize the communications between terrorists by attacking the most critical individuals in the networks [48].

People and contacts between them in a real society are represented as nodes and links of a graph in order to study the effect of epidemics in real social network [56, 69]. In order to prevent the spread of infectious diseases in real social networks, different strategies were presented for targeted vaccinations since random mass vaccinations are expensive [56, 69]. However, the optimal vaccination strategy is to find the critical nodes and vaccinate them to minimize the pairwise connectivity between people in a society [8], assuming that higher pairwise connectivity cause faster outbreak.

Biological organisms, such as bacteria and viruses, are sets of interconnected proteins

that interact with each other to form a protein-interaction network. These networks can be represented by a graph where nodes are proteins and edges are interactions between them. Protein structure and interactions are widely studied in biology [61, 63, 84, 87]. Identifying critical nodes (proteins), which maintain connectivity between all proteins, can provide useful information for many biological applications. For instance, in rational drug design [47, 53], these critical proteins need to be targeted to destroy and neutralize the corresponding harmful organism, resulting, from a pharmaceutical perspective, in a therapeutic benefit for the patient. This is what V. Tomaino et al. [75] investigated with the purpose of destroying aggressive cancer cells. To do so, they identified critical nodes in the human protein interaction network using the CNDP variant. Likewise, V. Boginski et al. [14] explored the identification of such proteins using the Cardinality Constrained-CNDP (CC-CNDP) variant. This work is a good application of the concept of critical nodes in computational biology.

Telecommunication networks such as the Internet, telephone networks, and computer networks can be represented as graphs, where each node is a terminal and links show the communications between terminals. In telecommunication networks, we want to prevent the spread of a virus or find some way to reduce as much as possible the communication within the network [9, 22]. Also, in network immunization [49, 50], where a graph representing contacts between people is given, only a given maximum number of persons can be vaccinated, and we aim at minimizing the propagation of the virus [21, 89].

The CNDP finds applications in the field of transportation engineering [33, 46]. Two particular examples are as follows. In general, for transportation networks, it is important to identify critical nodes in order to ensure they operate reliably for transporting people and goods throughout the network. Further, in planning for emergency evacuations, identifying the critical nodes of the transportation network is crucial. The reason is two-fold. First, knowledge of the critical nodes will help in planning the allocation of resources during the evacuation. Secondly, in the aftermath of a disaster they will help in re-establishing critical traffic routes.

The problem can be seen as a multicommodity version of the interdiction problems pioneered by Wollmer [85]. The problems deal with deleting arcs in order to minimize the maximum amount of flow that can be shipped through a network, a finite resource budget being allocated for arc deletion operations. The basic interdiction models deal with a single commodity source-sink flow on a directed capacitated network. Wood [86] also considers extensions to undirected graphs and multicommodity flows. Recently Smith and Lim [74] specifically tackle multicommodity interdiction models.

Myung and Kim [57] tackle the problem of deleting a limited number of edges from an undirected graph in order to minimize the weighted number of connections guaranteed in the residual graph.

Applications of the CNDP are not limited to what has been mentioned above, and there are many other applications in several other areas including the analysis of complex networks

[15, 34], security/vulnerability issues in networks [29, 58, 72, 73, 82], homeland security [16, 39, 44], energy [68], etc.

For problems based on the deletion of edges, a variety of variants have been defined and studied in the literature, such as the graph partitioning problem [17, 35, 62], the minimum k-cut problem [38, 41], the multicut problem [24, 37, 40], the multiway cut problem [18, 19, 26], the multi-multiway cut problem [12], etc.

A class of the CNDP that take into account the actual pairwise distances between nodes is so-called Distance-based Critical Node Detection Problem (D-CNDP) as introduced in [82]. The authors in [82] identify five different important classes of D-CNDP in which the objectives are to:

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

(ii) Minimize the Harary index;

(iii) Minimize the sum of power functions of distances;

(iv) Maximize the generalized Wiener index;

(v) Maximize the distance between two given nodes.

They propose the recursive formulation which is generic enough to handle all of the D-CNDP classes mentioned above. In [43], the authors propose a complementary mixed integer programming formulation and a Benders approach for the distance-based connectivity objective (iv) above. In [4] and [67], the authors propose new integer programming formulations as well as a heuristic approach for the first distance-based connectivity objective (i). Complexity analyses of the D-CNDP connectivity objectives (i), (ii) and (iv) above are presented in [7] for graphs with special structures, such as trees, paths and series–parallel graphs. The authors then proposed dynamic programming algorithms for polynomially and pseudo-polynomially solvable cases.

Connectivity Measures

The choice of the connectivity measure is of course a central element in a critical node or edge detection problem. Indeed, several different connectivity measures have been proposed in the literature. These measures can be categorized into two classes depending on the context of the problem that is being solved. The measures from the first class are mainly associated with network flow problems, in particular shortest path problems, maximum flow problems, or minimum cost flow problems. The logic behind these measures is that a network gets disconnected when it starts losing its ability to send flow between a predefined set of node pairs, or simply when traversing the network becomes too expensive [23, 85]. For these cases, the critical elements are the ones whose deletion results in the maximum increase of the

shortest paths or, consequently, the maximum decrease of the flow capacity between the predefined node pairs. These kinds of measures are commonly used in the context of network interdiction [20, 45, 55, 74], and are generally designed to tackle arc interdiction problems (detecting critical arcs).

On the other hand, the measures of the second class are mostly associated with topological characteristics of the network. Among this class, the most common measures are: the total number of pairwise connections (i.e., the total number of node pairs that are connected in the network by at least one path[1]) [8, 27], the total weighted pairwise connectivity (i.e., a weighted sum of the pairwise connections) [8, 27], the size of the largest connected[2] component (i.e., the number of nodes that belong to the largest maximal connected subgraph of a graph G) [59, 70], the total number of connected components [3, 70], average shortest path value [25], and the diameter [3]. Figure 1.1 provides an example of the different optimal solutions that are found depending on the measure that is chosen. For this example we assume that the goal is to identify the most critical node among the network. Note that if the connectivity measure is set to be the length of the shortest path between nodes 1 and 5, the critical node is node 3. On the other hand, if the measure is the size of the largest component, the critical node is node 4. And finally, if the measure is the total number of components, the critical node is node 6.

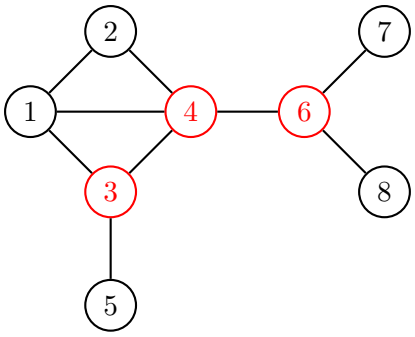

Figure 1.1: A graph to illustrate optimal solutions for different connectivity measures.

In this dissertation we consider the pairwise connectivity between nodes, formalized in [8]. The pairwise connectivity of a graph $G = (V, E)$ is defined as the number of pairs of nodes belonging to the same connected component.

This can be naturally generalized to the total *cost* of the pairs of nodes that belong to the same component, if connection costs are assigned for each pair of nodes.

[1] A *path* in a graph $G = (V, E)$ is a sequence of nodes $\{v_1, v_2, \ldots, v_k\}$ such that each pair (v_i, v_{i+1}) is an edge in E.

[2] A graph G is said to be *connected* if there is a path between any two distinct nodes. Otherwise, graph G is *disconnected*.

Computational Complexity

From the complexity point of view, the decision version[3] of the CNDP has been proven to be NP-complete on general graphs [8, 10, 29, 31, 54, 59, 71, 73] and often also remain NP-complete even on some special classes of graphs [2, 13, 27, 51, 70, 71, 73]. The recognition version of the CNDP with unit connection costs and node weights is proven to be NP-complete on general graphs by Arulselvan et al. [8] through a polynomial reduction from the Maximum Independent Set[4] problem, known to be NP-complete [36]. Polynomial algorithms are instead available only for some particular cases, which are usually limited to graphs with bounded treewidth, in particular trees and series-parallel graphs [2, 7, 13, 27, 51, 70]. Di Summa et al. [27] proved that the CNDP is also NP-complete on trees for the total weighted pairwise connectivity measures using a reduction from the multicut in trees problem [37]. They also showed that when the pairwise connection costs are set to be one (i.e., when the connectivity measure is replaced with the sum of the total number of pairwise connections), the problem can be solved in polynomial time using a dynamic programming approach. Moreover, Shen and Smith [25] proved that the CNDP is polynomially solvable in trees and series–parallel graphs for the cases when the deletion costs of the nodes are set to be one and the objective is either minimizing the size of the largest component or maximizing the number of residual components. They also propose a dynamic programming scheme for solving the CNDP in these cases. Most variants of D-CNDP are NP-hard [82]. However, some special cases of D-CNDP admit polynomial-time algorithms [7].

Solution Methodologies

Despite the difficulty of the problem, many approaches have been explored in the literature to solve it, using: dynamic programming [2, 13, 27, 51, 70] and integer linear programming [8, 10, 28, 30, 59, 71, 73], to provide exact solutions. To provide approximated solutions with performance guarantee, a variety of methods have been considered, including heuristic algorithms [1, 5, 6, 8, 9, 58, 64, 65, 66, 78], polynomial-time approximation algorithms [11, 13], particularly rounding-based approximation approaches [32, 73, 79, 80, 81], stochastic search algorithms [10, 77], fixed-parameter tractable algorithms [13, 54], polynomial pseudo-approximation algorithms [29]. Recently, solving frameworks considering different connectivity metrics together have been developed. This is the case for [83], where the devel-

[3]Decision problems are the problems for which the answer is either "YES" or "NO". We say that a decision problem is NP-complete if: (i) it is in NP, i.e., it has a polynomial-time verification algorithm, and (ii) all problems in NP are reducible to it in polynomial time. A decision problem X is polynomially reducible to a decision problem Y if there exists a polynomial-time reduction A such that, for any instance $I \in X$ we have $A(I) \in Y$. If only the second condition holds, then X is said to be NP-hard.

[4]In a graph $G = (V, E)$, an independent set is a set of vertices $S \subseteq V$ such that there is no edge between any two of them.

oped unifying integer programming framework takes into account four connectivity metrics. Similarly, authors in [5] presented an efficient evolutionary framework for solving different variants of the CNDP. The framework is potentially adaptable to deal with different variants of the CNDP considering different connectivity metrics. We refer the interested reader to the survey [52] for a detailed overview.

1.1 The problems that we study

According to the pairwise connectivity measure mentioned above, the Critical Node Detection Problem (CNDP) is formally stated as follows:

Problem 1 (CNDP)**.** *Given an undirected graph $G = (V, E)$, a weight $w_v \geq 0$ for every $v \in V$, a connection cost $c_{uv} \geq 0$ for all $u, v \in V$, and a weight limit $W \geq 0$, find $S \subseteq V$ such that the total weight of the nodes in S is at most W and the total cost of the pairs of nodes that are connected in $G - S$ is minimized.*

Here $G - S$ denotes the graph obtained after removing from G the nodes in S, i.e., the subgraph of G induced by $V \setminus S$. Furthermore, in the above problem as well in all variants considered below, the connection costs will be always implicitly assumed to be symmetric (i.e., $c_{uv} = c_{vu}$ for every $u, v \in V$) and to satisfy $c_{vv} = 0$ for every $v \in V$.

For this and the other problems that we study, we are particularly interested in the case in which G is a tree[5]. Already under this assumption, CNDP is NP-hard, even if $w_v = 1$ for every $v \in V$ and $c_{uv} \in \{0, 1\}$ for every $u, v \in V$ [27]. However, still assuming that G is a tree, the problem admits a polynomial-time algorithm if the c_{uv}'s are all equal to 1 (with no restriction on the w_v's) [27].

In this work we further investigate the complexity of CNDP on a tree, and consider some natural variants in which edges or both nodes and edges can be removed from the graph. Indeed, most of the literature seems to focus on problems where only nodes or edges can be deleted. (See, e.g., [83] for a discussion about this aspect.) Problems where both nodes and edges can be removed subject to a joint weight limit have been also considered in [83], where a general integer programming framework is presented for several types of connectivity measures. A different version of this problem has been studied in [42].

In order to simplify the description of the problems that we analyze, it is worthwhile to introduce some simple notation. Given an undirected graph $G = (V, E)$ and a subset $S \subseteq V \cup E$ of nodes and/or edges of G, $G - S$ will denote the graph obtained after removing from G the elements in S. More formally, $G - S = (V', E')$ with $V' = V \setminus S$ and $E' = \{uv \in E \setminus S : u, v \in V'\}$. If connection costs c_{uv} are given for all $u, v \in V$, we use the shorthand $c(G - S)$ to denote the total cost of the pairs of nodes that are connected in $G - S$. When weights w_v

[5] A *tree* $T(V, E)$ is a graph with no cycle, where any two nodes are connected by exactly one path. If a node is designated as root, the tree is called a rooted tree.

and/or w_e are given for all nodes $v \in V$ and/or all edges $e \in E$, $w(S)$ will denote the total weight of the elements in S. (Note that there is no confusion in denoting both node and edge weights with a similar symbol, i.e., w_v and w_e, as the different nature of the subscript —a node or an edge— removes any ambiguity. In other words, w can be viewed as a function of the form $w : V \cup E \to \mathbb{R}_+$.)

The above notation allows us to restate CNDP slightly more compactly:

Problem 2 (CNDP, restated)**.** *Given an undirected graph $G = (V, E)$, a weight $w_v \geq 0$ for every $v \in V$, a connection cost $c_{uv} \geq 0$ for all $u, v \in V$, and a weight limit $W \geq 0$, find $S \subseteq V$ such that $w(S) \leq W$ and $c(G - S)$ is minimized.*

As already mentioned, we are interested in some variants of CNDP (mainly on trees) in which edges or both nodes and edges can be removed. The variants that we analyze are the following:

- the Critical Edge Detection Problem (CEDP), which is formulated as CNDP, except that edges have to be removed instead of nodes;
- the Critical Node/Edge Detection Problem with a single weight limit (CNEDP-1), where a cumulative weight limit for the removal of nodes and edges is given;
- the Critical Node/Edge Detection Problem with two weight limits (CNEDP-2), where two separate weight limits are assigned for nodes and edges.

These problems are formalized below:

Problem 3 (CEDP)**.** *Given an undirected graph $G = (V, E)$, a weight $w_e \geq 0$ for every $e \in E$, a connection cost $c_{uv} \geq 0$ for all $u, v \in V$, and a weight limit $W \geq 0$, find $S \subseteq E$ such that $w(S) \leq W$ and $c(G - S)$ is minimized.*

Problem 4 (CNEDP-1)**.** *Given an undirected graph $G = (V, E)$, a weight $w_v \geq 0$ for every $v \in V$, a weight $w_e \geq 0$ for every $e \in E$, a connection cost $c_{uv} \geq 0$ for all $u, v \in V$, and a weight limit $W \geq 0$, find $S \subseteq V \cup E$ such that $w(S) \leq W$ and $c(G - S)$ is minimized.*

Problem 5 (CNEDP-2)**.** *Given an undirected graph $G = (V, E)$, a weight $w_v \geq 0$ for every $v \in V$, a weight $w_e \geq 0$ for every $e \in E$, a connection cost $c_{uv} \geq 0$ for all $u, v \in V$, and weight limits $W_V, W_E \geq 0$, find $S \subseteq V \cup E$ such that $w(S \cap V) \leq W_V$, $w(S \cap E) \leq W_E$, and $c(G - S)$ is minimized.*

We remark that CNDP and CEDP are special cases of each of CNEDP-1 and CNEDP-2. Indeed, an instance of CNDP (respectively, CEDP) can be reduced to an instance of CNEDP-1 by giving weight $W + 1$ to all edges (respectively, nodes) in order to forbid their removal. Furthermore, an instance of CNDP (respectively, CEDP) can be reduced to an instance of CNEDP-2 by setting $W_V = W$ and $W_E = 0$ (respectively, $W_E = W$ and $W_V = 0$).

1.2 Outline of the thesis and our contributions

The outline of this thesis is organized as follows.

In Chapter 2, we look at the problem of identifying important nodes in a path graph whose removal optimizes the connectedness of the given path. We analyze four variants of the critical node detection problem on paths and present a closed-form solution for the objective function's optimality.

In Chapter 3, we address the problem of detecting critical edges of a graph whose deletion optimally deteriorates the connectivity of the given graph. We present dynamic programming algorithms for the case with unit costs and unit edge weights as well as the case with unit costs and general edge weights and show that both algorithms can be executed in polynomial time. Then we move to the problem of detecting critical nodes and edges simultaneously by considering the case when a joint weight limit on the removal of nodes and edges is given and also given two separate weight limits for nodes and edges. We will see that CNEDP-1 can be solved in polynomial time when the underlying graph is a tree and the connection costs are all unitary, while for CNEDP-2 we have a polynomial-time algorithm only if we further assume that the node and edge weights are all equal to 1 (Proposition 3.5.2). On the contrary, we show that if the connection costs are unitary but the node and edge weights are general nonnegative numbers, CNEDP-2 is NP-hard to approximate within any factor, even when G is a path (Proposition 3.5.1).

In Chapter 4, we present a different approach to solve the CEDP on a tree. The idea is to subdivide the graph by inserting a new node in each edge and to remove only the newly created nodes instead of edges and the same complexity (see Chapter 3) is obtained.

Chapter 5 describes the NP-hardness results for the CEDP, CNEDP-1, and CNEDP-2 under the assumptions that the node/edge weights are unitary and the connection costs are 0/1 (Observation 12). We also present polynomial time algorithms for the different cases that we analyze, in which the connection costs are 0/1 and have the special structure: there exists $I \subseteq V$ such that $c_{uv} = 1$ if $u, v \in I$ and $u \neq v$, and $c_{uv} = 0$ otherwise. We call this special structure as square 0/1 connection costs.

In Chapter 6, we study our problems when the number of leaves of the tree is a constant. We first show that if the tree has exactly two leaves (i.e., a path) with arbitrary 0/1 connection costs and general nonnegative node weights, the CNDP can be solved in polynomial time in this situation. We then show more generally that CNDP, CEDP, and CNEDP-1 on a tree with arbitrary 0/1 connection costs and general nonnegative node/edge weights can be solved in polynomial time under this assumption (Proposition 14). The same result holds for CNEDP-2, but only if the node and edge weights are assumed to be unitary (Proposition 15). Indeed, we recall that without this further restriction CNEDP-2 is NP-hard already on a path (even with unit connection costs), i.e., on a tree with exactly two leaves (Proposition 3.5.1).

In Chapter 7, we propose new mathematical formulations based on the linear integer

programming (IP) model for finding exact solutions of the CNDP introduced by [8], which we define restricted CNDP. After that we derive IP models for the CEDP, CNEDP-1, and CNEDP-2 from the proposed restricted CNDP which provides optimal solutions.

We then focus on an extension of the critical node/edge detection problem in Chapter 8 where the distances between node pairs affect the objective function. We analyze some natural versions of Distance-based Critical Node/Edge Detection Problem and present dynamic programming algorithms whose complexities are polynomial. For the distance-based critical edge detection problem, we additionally propose an integer programming formulation.

Chapter 9 concludes this thesis with final observations. It also provides some open issues that remain to be covered.

Note that the polynomial-time algorithms that we obtain in this thesis take inspiration from the dynamic programming strategies proposed in [27] for CNDP on a tree with unit connection costs and in [7] for some distance-based versions of CNDP on a tree.

Chapter 2

Critical Node Detection Problem over a path

In this chapter we study the Critical Node Detection Problem (CNDP) on a path which can be seen as a very special case of a tree. Given a path graph $P = (V, E)$ with $|V| = n$ nodes and an integer K, the CNDP on a path seeks to find a set $S \subseteq V$ of at most $K \leq n$ nodes, the deletion of which minimizes pairwise connectivity in the remaining path graph $P(V \setminus S)$. We will tackle the so-called Distance-based Critical Node Detection Problem (D-CNDP) over paths as described in [7]. The NP-completeness of D-CNDP over paths for some specific distance functions has been already established [7]. We look at some particular versions of CNDP on paths that fall into the D-CNDP on paths and provide closed-form solution for optimality of the objective function. We note that these versions of D-CNDP on general trees will be studied in chapter 8. The variants that we analyze in this chapter are the following:

1. Minimizing the number of node-pairs connected by a path with upper bound of length at most L.
2. Minimizing the number of node-pairs connected by a path with lower bound of length at least L.
3. Minimizing the number of node-pairs connected by a path of length precisely L.
4. Minimizing the number of node-pairs connected by a path of length between L_1 and L_2 where L_1 and L_2 are the lower and upper bound respectively.

It can be easily seen that the CNDP with $L = 1$ on a graph $G = (V, E)$, which aims to minimize the number of edges that survive after having removed at most K nodes, is NP-complete through a reduction from Vertex cover. Indeed, deciding whether there is a vertex cover of size at most K in a graph is equivalent to verifying whether the optimal value of the CNDP with $L = 1$ is zero.

The term *balanced solution* is an important concept that we will utilize in the following statements. We call the solution balanced if the difference between the length of the shortest and longest components is either 0 or 1.

Before deriving the exact solution of the objective function for the four variants introduced above, we first present the following result without any lower or upper bound on the length of the connecting path.

Proposition 6. *For the CNDP on a path without any lower or upper bound, the optimal solutions are the balanced solutions.*

Proof. Let $P = (V, E)$ be a path graph with $|V| = n$ nodes and let α be the average number of nodes of the sub-paths obtain after removing K nodes, where $\alpha = \frac{|V|-K}{K+1}$.

Assume that we are given an optimal solution after removing K nodes. Suppose by contradiction that in the given optimal solution, not all the sub-paths have length $\lfloor\alpha\rfloor$ or $\lceil\alpha\rceil$. Since the solution is unbalanced, there are at least two sub-paths in this solution in which one of the component has length at most $\lfloor\alpha\rfloor - 1$ and another component has length at least $\lceil\alpha\rceil + 1$. Let l_1 and l_2 be the number of nodes of the shortest and the longest components respectively, i.e., $l_1 \leq \lfloor\alpha\rfloor - 1$ and $l_2 \geq \lceil\alpha\rceil + 1$ (see Figure 2.1). Now we construct a new solution by adding one node in the shortest component, i.e., l_1+1 and by subtracting one node in the longest component, i.e., $l_2 - 1$ and the length of the other components are unchanged (see Figure 2.2). We want to show that this new solution is better than the given solution.

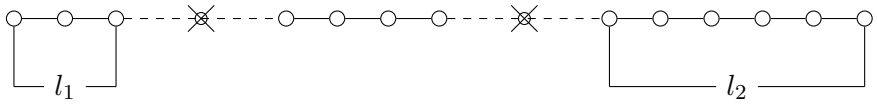

Figure 2.1: Example of a solution after removing $K = 2$ nodes.

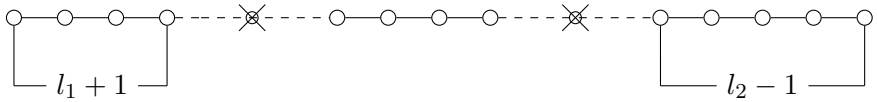

Figure 2.2: Modification of the solution of Figure 2.1.

So the claim is

$$\binom{l_1+1}{2} + \binom{l_2-1}{2} < \binom{l_1}{2} + \binom{l_2}{2}, \quad \text{where} \quad l_2 - l_1 \geq 2$$

$$\iff (l_1+1)l_1 + (l_2-1)(l_2-2) < l_1(l_1-1) + l_2(l_2-1)$$

$$\iff l_1(l_1+1-l_1+1) < (l_2-1)(l_2-l_2+2)$$

$$\iff 2l_1 < 2(l_2 - 1)$$

$$\iff l_2 - l_1 > 1$$

which is true by assumption because we are assuming $l_2 - l_1 \geq 2$, and we will continue this perturbation until we have $K+1$ components of size $\lfloor \alpha \rfloor$ and $\lceil \alpha \rceil$, and this shows that every optimal solution is balanced. Observe that all balanced solutions have the same cost and hence they are equivalent. Since all balanced solutions are equivalent, this proves that all balanced solutions are optimal.

□

We now state the result when the graph is a path and the problem is to remove K nodes in order to minimize the number of pairs that are connected by a path of length at most L, for some given natural number L.

Proposition 7. *For the CNDP on a path with upper bound L, any solution that removes K nodes and creates $K+1$ components with cardinality at least L is optimal. If no such a solution exists, then any solution that removes K nodes and creates balanced components is optimal.*

Proof. First we prove the second part of the statement which means there is no way to make all the components with cardinality at least L and in this case we want to prove that balanced solutions are optimal.

Assume by contradiction that we are given an optimal solution which is not balanced. Since we are assuming that there is no way to make all the components with cardinality at least L, there is at least one sub-path with cardinality at most $L-1$. Let A be the shortest component and B be the longest component of the solution. With a slight abuse of notation we will denote by A and B also the number of nodes in the component. Then $A \leq L-1$ and $B - A \geq 2$ because the solution is not balanced. Now we do the perturbation by adding one node in the shortest component and by subtracting one node in the longest component and by keeping the cardinality of all other components unchanged. Now if $B > L$ and we subtract a node from B, then we lose L connections, while if $B \leq L$ and we subtract a node from B, then we lose $(B-1)$ connections because we have to count only the paths of length $\leq L$. Since $A \leq L-1$, if we add an extra node to A, then we have A new connections. Thus we have the following two cases when we do the perturbation.

Case 1: If $B > L$, then the improvement is < 0.

Case 2: If $B \leq L$, then the improvement is $A - B + 1$ which becomes ≤ -1 because we are assuming that $B - A \geq 2$.

So with the property $A \leq L-1$, we have always a negative improvement (case 1 and 2), i.e., we reduce the objective function and this proves that any unbalanced solution is not

optimal. Since all balanced solutions are equivalent, this proves that all balanced solutions are optimal.

Now we prove the first part of the statement which means that it is possible to make all the sub-paths with cardinality at least L. In this case still it is possible that in the solution the shortest component is smaller than $L-1$ and we know by the above argument that this solution is not optimal.

In the following we prove that all the solutions that creates $K+1$ sub-paths with cardinality at least L are equivalent.

When we remove K nodes from the path, we create $K+1$ components. Suppose t_i is the number of edges of every sub-path. Now if we take a path with $t_i \geq L-1$ edges, then the contribution is given by $L(t_i - L + 1) + \sum_{K=1}^{L-1} K$, because in a component of $(t+1)$ nodes, we have $(t+1-L)$ nodes which gives a contribution of length L and the remaining L nodes gives the contribution of $(L-1), (L-2), \ldots, 1$, respectively, which is given by $\sum_{K=1}^{L-1} K$. In the last component, the number of edges is given by $t_{K+1} = n - 1 - \sum_{i=1}^{K} t_i - 2K$, because we have $n-1$ edges initially, then we have to discard all the edges of the sub-paths which is given by $\sum_{i=1}^{K} t_i$ and when we are removing K nodes, we are deleting $2K$ edges, since we are assuming that we are creating $K+1$ components that means we are not removing two adjacent nodes. So the contribution of the $(K+1)$th component is given by $L(n-1-\sum_{i=1}^{K} t_i - 2K) + \sum_{K=1}^{L-1} K$. Hence the total contribution is

$$\sum_{i=1}^{K} L(t_i - L + 1) + \sum_{K=1}^{L-1} K + L(n - 1 - \sum_{i=1}^{K} t_i - 2K) + \sum_{K=1}^{L-1} K = 2\sum_{K=1}^{L-1} K + L(n - L - 2K)$$

which is independent of t_i's, that means it does not matter what is the length of the sub-paths as long as their cardinality is at least L.

Now we show that all the solutions that create all components with cardinality at least L are optimal. Define S^* as the set of solutions which have the property that all components have at least L nodes. Since there is at least one solution with all the components with cardinality at least L, the set S^* is nonempty. Take a solution in S^*. This solution has the property that the total number of nodes in the surviving components is at least $(L-1)(K+1)$ as after removing K nodes we have $(K+1)$ components and each of cardinality at least $(L-1)$. Let's call again A and B be the shortest component and the longest component respectively in S^*. Take any solution which is not in S^*, which means that it has at least one component whose cardinality must be smaller than $L-1$, so that $A \leq L-2$. We claim that the cardinality of the longest component is at least L, i.e., $B \geq L$. If this is not true, then $B \leq L-1$. In this case the total number of nodes in the surviving components is at most $(L-2) + (L-1)K$, which is at least one unit smaller than $(L-1)(K+1)$ and this is a contradiction because we know that the total number of nodes can not be changed as we always remove K nodes. So any solution which is not in S^* has the property that $A \leq L-2$ and $B \geq L$ and this means that the solution is not balanced. Since the solution is not balanced, we apply the perturbation

and by the previous arguments (case 1 and case 2) we know that when we do the perturbation we always improve. After the perturbation either we find a solution which is in S^* and we are done, or we find a solution which has still the property that $A \leq L-2$ and $B \geq L$ and we will continue this perturbation until the solution is balanced. When we get a balanced solution, we are in S^* and this proves that the set S^* is optimal.

□

We note that Proposition 6 is a special case of Proposition 7 because Proposition 7 coincides with Proposition 6 when we put $L = n$.

The following result holds for the case when the graph is a path and the problem calls for minimizing the number of node-pairs still connected by a path of length at least L, for some given natural number L, surviving in a path after having removed at most K nodes..

Proposition 8. *For the CNDP on a path with lower bound L, any solution that removes K nodes and creates $K+1$ components with cardinality at most L is optimal. If no such a solution exists, then any solution that removes K nodes and creates balanced components is optimal.*

Proof. First consider that we can make all the components with cardinality at most L and in this case all the solutions are equivalent and hence optimal, because the objective value is 0 and in this incident, it does not matter whether the solution is balanced or not as long as all the components have cardinality at most L.

Now we prove the second part of the statement which means there is no way to make all the components with cardinality at most L and in this case we want to prove that balanced solutions are optimal.

Assume by contradiction that we are given an optimal solution which is not balanced. Since we are assuming that there is no way to make all the components with cardinality at most L, there is at least one sub-path with cardinality greater than L. Let A be the shortest component and B be the longest component of the solution. With a slight abuse of notation we will denote by A and B also the number of nodes in the component. Then $B > L$ and $A \leq B-2$ because the solution is not balanced. Now we do the perturbation by adding one node in the shortest component and by subtracting one node in the longest component and by keeping the cardinality of all other components unchanged. Now if $A \geq L-1$ and we add an extra node to A, then we have $A-L+1$ new connections while if $A < L-1$ and we add an extra node to A, in this case we create 0 connection, because we have to count only the paths of length $\geq L$. Since $B > L$ and we subtract a node from B, we lose $B-L$ connections. Thus we have two cases when we do the perturbation.

Case 1: If $A \geq L-1$, then the improvement is $A-B+1$ which becomes ≤ -1 because we are assuming that $A \leq B-2$.

Case 2: If $A < L-1$, then the improvement is $L-B$ which becomes ≤ -1 because $B > L$.

So when we have at least one sub-path with cardinality at least L, we always get a better solution and this proves that any unbalanced solution is not optimal. Since all balanced solutions are equivalent, this proves that all balanced solutions are optimal.

□

When the graph is a path, and we want to minimize the number of node-pairs still connected by a path of precise length L, for some given natural number L, surviving in a path after having removed at most K nodes, the following result holds.

Proposition 9. *For the CNDP on a path of precise length L, any solution that removes K nodes and creates $K+1$ components with cardinality at least L is optimal. Otherwise, any solution that creates $K+1$ components with cardinality at most L is optimal. If no such a solution exists, then any solution that removes K nodes and creates balanced components is optimal.*

Proof. Assume by contradiction that we are given an optimal solution which is not balanced. We are assuming that there is no way to make all the components with cardinality at least L or all the components with cardinality at most L. Then there is at least one sub-path with cardinality less than L and at least one sub-path with cardinality greater than L. Let A be the shortest component and B be the longest component of the solution. With a slight abuse of notation we will denote by A and B also the number of nodes in the component. Then $A < L$, $B > L$ and $A \leq B - 2$ because the solution is not balanced. Now we do the perturbation by adding one node in the shortest component and by subtracting one node in the longest component and by keeping the cardinality of all other components unchanged. Since $A < L$, when we add an extra node to A, we create 0 connection, because we have to count only the paths of length precisely L. Again since $B > L$, when we subtract a node from B, we lose 1 connection. Thus we have the following case when we do the perturbation.

Case 1: If $A < L$ and $B > L$, then the improvement is -1.

In the following we prove that all the solutions that creates $K+1$ sub-paths with cardinality at least L are equivalent.

When we remove K nodes from the path, we create $K+1$ components. Suppose t_i is the number of edges of every sub-path. Now if we take a path with $t_i \geq L-1$ edges, then the contribution is given by $(t_i - L + 1)$, because in a component of $(t+1)$ nodes, we have $(t_i + 1 - L)$ nodes which gives a contribution of length exactly L. In the last component, the number of edges is given by $t_{K+1} = n - 1 - \sum_{i=1}^{K} t_i - 2K$, because we have $n-1$ edges initially, then we have to discard all the edges of the sub-paths which is given by $\sum_{i=1}^{K} t_i$ and when we are removing K nodes, we are deleting $2K$ edges, since we are assuming that we are creating $K+1$ components that means we are not removing two adjacent nodes. So the contribution

of the $(K+1)$th component is given by $(n-1-\sum_{i=1}^{K} t_i - 2K)$. Hence the total contribution is

$$\sum_{i=1}^{K}(t_i - L + 1) + (n - 1 - \sum_{i=1}^{K} t_i - 2K) = n - L - 2K$$

which is independent of t_i's, that means it does not matter what is the length of the sub-paths as long as their cardinality is at least L and this proves that all the solutions that creates long components are all equivalent.

Now we show that all the solutions that create all components with cardinality at least L are optimal. Define S^* as the set of solutions which have the property that all components have at least L nodes. Since there is at least one solution with all the components with cardinality at least L, the set S^* is nonempty. Take a solution in S^*. This solution has the property that the total number of nodes in the surviving components is at least $(L-1)(K+1)$ as after removing K nodes we have $(K+1)$ components and each of cardinality at least $(L-1)$. Let's call again A and B be the shortest component and the longest component respectively in S^*. Take any solution which is not in S^*, which means that it has at least one component whose cardinality must be smaller than $L-1$, so that $A \leq L-2$. We claim that the cardinality of the longest component is at least L, i.e., $B \geq L$. If this is not true, then $B \leq L-1$. In this case the total number of nodes in the surviving components is at most $(L-2)+(L-1)K$, which is at least one unit smaller than $(L-1)(K+1)$ and this is a contradiction because we know that the total number of nodes can not be changed as we always remove K nodes. So any solution which is not in S^* has the property that $A \leq L-2$ and $B \geq L$ and this means that the solution is not balanced. Since the solution is not balanced, we apply the perturbation and by the previous arguments (case 1) we know that when we do the perturbation we always improve. After the perturbation either we find a solution which is in S^* and we are done, or we find a solution which has still the property that $A \leq L-2$ and $B \geq L$ and we will continue this perturbation until the solution is balanced. When we get a balanced solution, we are in S^* and this proves that the set S^* is optimal.

Similarly, if we can make component in such a way that the cardinality of all components is shorter than L, then all the solutions are equivalent and hence optimal, because the objective value is 0 and in this incident, it does not matter whether the solution is balanced or not as long as all the components have cardinality at most L.

Otherwise, if we can not make solution in which all the components have cardinality at least L or at most L, then obviously we are in case 1, because case 2 is not possible and in this case we always have a negative improvement, i.e., we reduce the objective function and this proves that any unbalanced solution is not optimal. Since all balanced solutions are equivalent, this proves that all balanced solutions are optimal.

□

In the following we state the result when the graph is a path and the problem is to remove

K nodes in order to minimize the number of pairs that are connected by a path of length at least L_1 and at most L_2.

Now we consider the case in which we have a lower bound L_1 and an upper bound L_2 and we consider the connections which are between L_1 and L_2.

Proposition 10. *For the CNDP on a path with lower bound L_1 and upper bound L_2, any solution that removes K nodes and creates $K+1$ components with cardinality at least L_2 is optimal. Otherwise, any solution that creates $K+1$ components with cardinality at most L_1 is optimal. If no such a solution exists, then any solution that removes K nodes and creates balanced components is optimal.*

Proof. Assume by contradiction that we are given an optimal solution which is not balanced. We are assuming that there is no way to make all the components with cardinality at least L_2 or all the components with cardinality at most L_1. Let A be the shortest component and B be the longest component of the solution. With a slight abuse of notation we will denote by A and B also the number of nodes in the component. Then $A < L_2$, $B > L_1$ and $A \leq B - 2$ because the solution is not balanced. Now we do the perturbation by adding one node in the shortest component and by subtracting one node in the longest component and by keeping the cardinality of all other components unchanged. Since we want to count the paths of length between L_1 and L_2, if $A < L_1 - 1$ and we add an extra node to A, then we have 0 new connection while if $L_1 - 1 \leq A < L_2$ and we add an extra node to A, in this case we create $A - L_1 + 1$ connections. Again if $L_1 < B < L_2$ and we subtract a node from B, then we lose $B - L_1$ connections while if $B > L_2$ and we subtract a node from B, then we lose $L_2 - L_1$ connections. Thus we have the following cases when we do the perturbation.

Case 1: If $A < L_1 - 1$ and $L_1 < B < L_2$, then the improvement is $L_1 - B$ which becomes ≤ -1 because $L_1 < B$.

Case 2: If $A < L_1 - 1$ and $B > L_2$, then the improvement is $L_1 - L_2$ which becomes ≤ -1 since $L_1 < L_2$.

Case 3: If $L_1 - 1 \leq A < L_2$ and $L_1 < B < L_2$, then the improvement is $A - B + 1$ which becomes ≤ -1 because we are assuming that $A \leq B - 2$.

Case 4: If $L_1 - 1 \leq A < L_2$ and $B > L_2$, then the improvement is $A - L_2 + 1$ which becomes ≤ -1 because $A < L_2$.

If we can make all the components with cardinality at most L_1, then all the solutions are equivalent and hence optimal, because the objective value is 0.

In the following first we prove that all the solutions that creates $K + 1$ sub-paths with cardinality at least L_2 are equivalent.

When we remove K nodes from the path, we create $K + 1$ components. Suppose t_i is the number of edges of every sub-path. Now if we take a path with $t_i \geq L_2 - 1$ edges, then

the contribution is given by $(L_2 - L_1 + 1)(t_i - L_2 + 1) + \sum_{K=1}^{L_2-L_1} K$, because in a component of $(t_i + 1)$ nodes, we have $(t_i + 1 - L_2)$ nodes which gives a contribution of $(L_2 - L_1 + 1)$ and the remaining $L_2 - 1$ nodes gives the contribution of $(L_2 - L_1), (L_2 - L_1 - 1), \ldots, 1$, respectively, which is given by $\sum_{K=1}^{L_2-L_1} K$. In the last component, the number of edges is given by $t_{K+1} = n - 1 - \sum_{i=1}^{K} t_i - 2K$, because we have $n - 1$ edges initially, then we have to discard all the edges of the sub-paths which is given by $\sum_{i=1}^{K} t_i$ and when we are removing K nodes, we are deleting $2K$ edges, since we are assuming that we are creating $K+1$ components that means we are not removing two adjacent nodes. So the contribution of the $(K + 1)$th component is given by $(L_2 - L_1 + 1)(n - 1 - \sum_{i=1}^{K} t_i - 2K) + \sum_{K=1}^{L_2-L_1} K$. Hence the total contribution is

$$\sum_{i=1}^{K}(L_2 - L_1 + 1)(t_i - L_2 + 1) + \sum_{K=1}^{L_2-L_1} K + (L_2 - L_1 + 1)(n - 1 - \sum_{i=1}^{K} t_i - 2K) + \sum_{K=1}^{L_2-L_1} K$$

$$= 2\sum_{K=1}^{L_2-L_1} K + (L_2 - L_1 + 1)(n - L_2 - 2K)$$

which is independent of t_i's, that means it does not matter what is the length of the sub-paths as long as their cardinality is at least L_2 and this proves that all the solutions that creates long components are all equivalent.

Now we show that all the solutions that create all components with cardinality at least L_2 are optimal. Define S^* is the set of solutions which has the property that all components have at least L_2 nodes. Since there is at least one solution with all the components with cardinality at least L_2, the set S^* is nonempty. Take a solution in S^*. This solution has the property that the total number of nodes in the surviving components is at least $(L_2 - 1)(K + 1)$ as after removing K nodes we have $(K + 1)$ components and each of cardinality at least $(L_2 - 1)$. Let's call again A and B be the shortest component and the longest component respectively in S^*. Take any solution which is not in S^*, which means that it has at least one component whose cardinality must be smaller than $L_2 - 1$, so that $A \leq L_2 - 2$. We claim that the cardinality of the longest component is at least L_2, i.e., $B \geq L_2$. If this is not true, then $B \leq L_2 - 1$. In this case the total number of nodes in the surviving components is at most $(L_2 - 2) + (L_2 - 1)K$, which is at least one unit smaller than $(L_2 - 1)(K + 1)$ and this is a contradiction because we know that the total number of nodes can not be changed as we always remove K nodes. So any solution which is not in S^* has the property that $A \leq L_2 - 2$ and $B \geq L_2$ and this means that the solution is not balanced. Since the solution is not balanced, we apply the perturbation and by the previous arguments (case 2 and case 4) we know that when we do the perturbation we always improve. After the perturbation either we find a solution which is in S^* and we are done, or we find a solution which has still the property that $A \leq L_2 - 2$ and $B \geq L_2$ and we will continue this perturbation until the solution is balanced. When we get a balanced solution, we are in S^* and this proves that the set S^* is optimal.

Otherwise, if we can not make solution in which all the components have cardinality at least L_2 or at most L_1, then we are in one of the four cases and in every case we always have a negative improvement, i.e., we reduce the objective function and this proves that any unbalanced solution is not optimal. Since all balanced solutions are equivalent, this proves that all balanced solutions are optimal.

□

We would like to mention that Propositions 7, 8, and 9 are all special instances of Proposition 10. By placing $L_1 = 0$ and $L_2 = L$, Proposition 7 can be deduced from Proposition 10. Similarly, we obtain the problem specified in Proposition 9 when we write $L_1 = L_2 = L$ in Proposition 10, and Proposition 10 corresponds with Proposition 8 when we put $L_1 = L$ and $L_2 = n$.

Chapter 3

Polynomial time algorithms of CEDP, CNEDP-1, and CNEDP-2 on a tree

In this chapter, we look at the critical edge detection problem (CEDP), which is the task of finding a set of K edges in a graph G whose removal reduces the number of connections between pairs of nodes in the residual graph. We investigate the CEDP over trees, generalizing the objective function and constraints to account for generic nonnegative costs of node connections and weights for edges to be deleted. We also investigate the complexity of the Critical Node/Edge Detection Problem with a single weight limit (CNEDP-1), where a cumulative weight limit for the removal of nodes and edges is given and the Critical Node/Edge Detection Problem with two weight limits (CNEDP-2), where two separate weight limits are assigned for nodes and edges.

In Section 3.1.1 we present a dynamic programming approach for the case with unit costs and unit edge weights whose complexity is polynomial. In Section 3.1.2 we deal with the case with unit costs and general edge weights. In Section 3.2 and Section 3.3 we present the case with unit costs and a weight limit for the removal of nodes and edges respectively. In Section 3.4 we show that CNEDP-1 can be solved in polynomial time when the connection costs are all unitary. In Section 3.5 we prove that CNEDP-2 with unit connection costs is $\mathcal{NP}$-complete even on a path, while CNEDP-2 admits a polynomial-time algorithm if the node and edge weights are fixed to 1 as well as the connection costs which is presented in Section 3.5.1.

3.1 Solving CEDP on a tree with unit connection costs

In this section we show how to solve CEDP on a tree in polynomial time when the connection costs are all unitary. We note that structural parts of our algorithmic framework are similar to those provided in [27, 51, 70].

Let $G = (V, E)$ be a tree with $|V| = n$, and let the input data be denoted as in Problem 3 (see the introduction), where the connection costs c_{uv} are defined as follows:

$$c_{uv} = \begin{cases} 1 & \text{if } u, v \text{ are in the same component of } G(E \setminus S) \\ 0 & \text{otherwise.} \end{cases} \tag{3.1}$$

Formally, CEDP's goal is to discover S whose objective function yields the smallest value.

Throughout the thesis in our all dynamic programs, we denote by T_a the subtree of the given tree $T(V, E)$ rooted at node $a \in V$. If a is not a leaf of T, we assume that an arbitrary order of its children is specified. If a has s children $a_1, \ldots, a_s$, for every $i \in \{1, \ldots, s\}$ we define $T_{a_{i,s}}$ as the subtree of T induced by $\{a\} \cup V(T_{a_i}) \cup \cdots \cup V(T_{a_s})$, where we denote by $V(H)$ the set of vertices of H for any given subtree H of T. Figure 3.1 shows an example of a tree T rooted at node a where subtree T_{a_2} contains nodes of the set $\{a_2, 3, 4, 5, 6, 7\}$, while subtree $T_{a_{3,4}}$ contains nodes of the set $\{a, a_3, 8, 9, 10, 11, 12, a_4, 13, 14, 15\}$. In our dynamic programming approaches, all recursions are based on traversing the tree in postorder (that is, from the leaves to the root) and from the right part of each tree level to the left part.

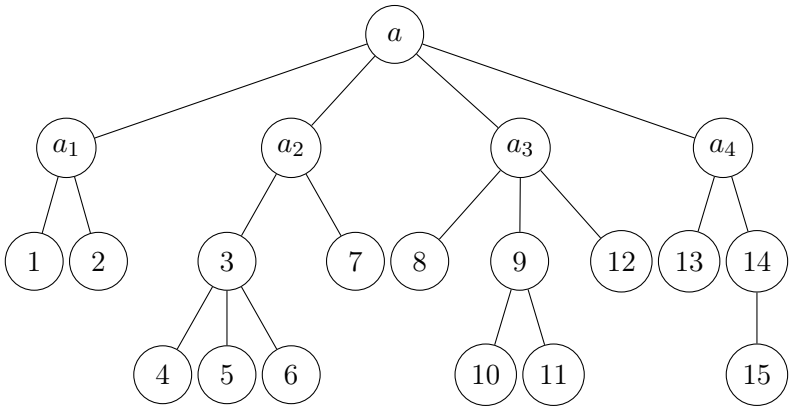

Figure 3.1: T_a is an example of a subtree in which node a has four children (i.e. $s = 4$).

In the next two subsections, we will look at unit weights and general nonnegative weights for the edges separately, since even though the latter subsumes the former, unitary weights have a faster running time.

3.1.1 The unit cost, unit weight case on trees

In this part, we show how to solve CEDP on trees when $c_{uv} = 1$ for all u, v and $w_v = 1$ for all $v \in E$. In this scenario, the goal is to reduce the number of paths left in a tree $T(V, E)$ after removing at most K edges.

To derive a dynamic programming algorithm, we will calculate recursively the following values:

- $F_a(m,k)$ = minimum number of connections that still exists in the subtree T_a when k edges are removed from T_a and m nodes (including a itself) of T_a are still connected to the root a.
- $G_{a_i}(m,k)$ = minimum number of connections that still exists in the subtree $T_{a_{i,s}} = a + T_{a_i} + \cdots + T_{a_s}$ when k edges are removed from $T_{a_{i,s}}$ and m nodes of the subtree are still connected to a.

We remark that for both functions, the number of nodes connected to the root will never be 0 (i.e. $m > 0$) because we never remove the root as the root is a node and we can not remove a node in the edge deletion problem. Furthermore, whenever the conditions in one of the above definitions cannot be satisfied, we set the value of the function to infinity.

The values of F_a and G_{a_i} are calculated in this order:

- we determine F_a for every leaf a;
- for a non-leaf node a, assuming that the $F_{a'}$ and $G_{a'_i}$ have been already found for all $a' \in V(T_a)$, we calculate $G_{a_s}, G_{a_{s-1}}, \ldots, G_{a_1}$, and then F_a.

At the end of the recursion, we can return the optimal value of the problem, assuming that the tree T is rooted at node 1, which is

$$\text{OPT} = \min\{F_1(m,K) : m = 1, \ldots, n\}.$$

As usual in dynamic programming, an optimal solution can be reconstructed by backtracking.

We now provide the explicit formulas and then a justification for each of them. For a non-leaf node $a \in V$, we have

$$F_a(m,k) = G_{a_1}(m,k), \tag{3.2}$$

while for every leaf a the formula is

$$F_a(m,k) = \begin{cases} 0 & \text{if } m = 1,\ k = 0, \\ \infty & \text{otherwise.} \end{cases} \tag{3.3}$$

For any non-leaf node $a \in V$ and $i < s$ (non-rightmost subtrees) we use the formula

$$G_{a_i}(m,k) = \begin{cases} \min\{F_{a_i}(p,0) + G_{a_{i+1}}(m-p,0) + p(m-p) : p = 0, \ldots, m\} & \text{if } k = 0, \\ \min\{\min\{F_{a_i}(p,q) + G_{a_{i+1}}(m, k-1-q) : p = 0, \ldots, \left|V(T_{a_i})\right|, \\ \qquad q = 0, \ldots, k-1\},\ \min\{F_{a_i}(p,q) + G_{a_{i+1}}(m-p, k-q) + p(m-p) : \\ \qquad p = 0, \ldots, m,\ q = 0, \ldots, k\}\} & \text{if } k > 0, \end{cases} \tag{3.4}$$

The initial conditions on each rightmost subtree T_{a_s} are calculated as follows::

$$G_{a_s}(m,k) = \begin{cases} \infty & \text{if } m=1,\ k=0, \\ \min\{F_{a_s}(p,k-1) : p=0,\ldots,|V(T_{a_s})|\} & \text{if } m=1,\ k>0, \\ F_{a_s}(m-1,k)+(m-1) & \text{if } m>1,\ k\geq 0. \end{cases} \tag{3.5}$$

We now give a justification for the above formulas. Equation (3.2) follows because $T_a = T_{a_{1,s}}$ for any non-leaf node $a \in V$.

Equation (3.3) handles the case of a one-node tree. Since $a \in V$ is a leaf, it is not possible to remove any edge ($k=0$) and only a is connected to itself ($m=1$) and the number of paths surviving in T_a is 0.

Recursion (3.4) can be interpreted as follows:

The case $k=0$ means that we are not removing any edge from $T_{a_{i,s}} = T_{a_i} + T_{a_{i+1,s}}$. Since we have to keep everything, we are not allowed to remove anything from the subtrees T_{a_i} and $T_{a_{i+1,s}}$. If a_i is connected to p nodes of T_{a_i}, then a is connected to $m-p$ nodes in $T_{a_{i+1,s}}$ and the paths passing through a are exactly $p(m-p)$. Hence by definition of F and G the minimum number of paths that survive in $T_{a_{i,s}}$ when we are not removing anything will be $G_{a_i}(m,0) = \min_p\{F_{a_i}(p,0) + G_{a_{i+1}}(m-p,0) + p(m-p)\}$.

The case $k>0$ means that we have to remove at least one edge. When the value of $G_{a_i}(m,k)$ is achieved from the expression $F_{a_i}(p,q) + G_{a_{i+1}}(m,k-1-q)$, we remove the edge e (which connects a to a_i). Expression $F_{a_i}(p,q)$ gives the minimum number of paths that survive in T_{a_i} when q edges are removed from T_{a_i} and p nodes of T_{a_i} are still connected to a_i. Since q edges have been removed from T_{a_i}, exactly $k-1-q$ edges must be removed from $T_{a_{i+1,s}}$. The minimum number of paths that survive in $T_{a_{i+1,s}}$ when $k-1-q$ edges are removed from $T_{a_{i+1,s}}$ is given by $G_{a_{i+1}}(m,k-1-q)$. Thus the expression $F_{a_i}(p,q)+G_{a_{i+1}}(m,k-1-q)$ gives the minimum number of paths that survive in $T_{a_{i,s}}$ when q edges are removed from T_{a_i} (and the other $k-1-q$ edges are removed from $T_{a_{i+1,s}}$) and p nodes of T_{a_i} are still connected to a_i. By taking the minimum over $p=0,\ldots,|V(T_{a_i})|$ and $q=0,\ldots,k-1$, we find the value of $G_{a_i}(m,k)$.

When the value of $G_{a_i}(m,k)$ is achieved from the expression $F_{a_i}(p,q) + G_{a_{i+1}}(m-p,k-q) + p(m-p)$, we are not removing the edge e. As above, expression $F_{a_i}(p,q)$ gives the minimum number of paths that survive in T_{a_i} when q edges are removed from T_{a_i} and p nodes of T_{a_i} are still connected to a_i. Since q edges have been removed from T_{a_i}, exactly $k-q$ edges must be removed from $T_{a_{i+1,s}}$ and since p nodes of T_{a_i} are still connected to a_i and thus to a, exactly $m-p$ nodes of $T_{a_{i+1,s}}$ must remain connected to a. The minimum number of paths that survive in $T_{a_{i+1,s}}$ when $k-q$ edges are removed from $T_{a_{i+1,s}}$ and $m-p$ nodes of $T_{a_{i+1,s}}$ are still connected to a is given by $G_{a_{i+1}}(m-p,k-q)$. Thus the expression $F_{a_i}(p,q)+G_{a_{i+1}}(m-p,k-q)$ gives the minimum number of paths that survive in T_{a_i} or $T_{a_{i+1,s}}$

when q edges are removed from T_{a_i} (and the other $k-q$ edges are removed from $T_{a_{i+1,s}}$) and p nodes of T_{a_i} are still connected to a_i, while $m-p$ nodes of $T_{a_{i+1,s}}$ are still connected to a. Now we have to add the paths connecting nodes of T_{a_i} to nodes of $T_{a_{i+1,s}}$, i.e. $p(m-p)$ paths. This gives expression $F_{a_i}(p,q) + G_{a_{i+1}}(m-p, k-q) + p(m-p)$ of recursion (3.4). By taking the minimum over $p = 0, \ldots, m$ and $q = 0, \ldots, k$, we find the value of $G_{a_i}(m,k)$.

More specifically, if $k = 0$, we have only one choice so that we have to keep the edge e (which connects a to a_i) because we are not removing any edge. While in the case $k > 0$, we have two possibilities that both are possible i.e., we can choose if we want to remove the edge e or we want to keep it.

For a justification of (3.5), recall that $T_{a_{s,s}} = a + T_{a_s}$. If $m = 1$ and $k > 0$, then we have to remove the edge between a and a_s and the other $k-1$ edges we have to be removed from the subtree T_{a_s} and the number of connections that survive are those in the subtree T_{a_s}. On the other hand if $m > 1$, then we can not remove the edge a to a_s and in this time we have to remove all the k edges inside the subtree T_{a_s}. Since m nodes are connected to a including a itself, in the subtree we will find the other $m-1$ nodes connected to a_s. Then we have to add all the connections of a to the nodes that are connected to a_s in the subtree.

We obtain the following result.

Proposition 3.1.1. *CEDP on a tree with unit connection costs and unit edge weights can be solved by recursion* (3.2)–(3.5) *in* $\mathcal{O}(n^3K^2)$ *time.*

Proof. For each node $a \in V$ there are at most $n+1 = \mathcal{O}(n)$ values for m and $K+1 = \mathcal{O}(K)$ values for k; this gives $\mathcal{O}(n^2K)$ values of F and G to compute. The heaviest computation is that of equation (3.4) that requires at most $\mathcal{O}(nK)$ steps. Hence in the worst case a number of operations bounded by $\mathcal{O}(n^3K^2)$ are required. □

3.1.2 The case with unit costs and arbitrary edge weights

Let $w_e \geq 0$ be arbitrary weights assigned to the edges $e \in E$. The CEDP problem in this case amounts to finding a subset S of edges with total weight $\sum_{e \in S} w_e$ not exceeding a given budget W such that the number of surviving paths after having removed the edge set S is minimized.

A dynamic programming algorithm, constructed in the same spirit as the one described in the preceding section, can be used to solve this case. The recursion uses two parameters, m and k, which represent the number of nodes connected to the root of a subtree and the number of paths that survive within that subtree, respectively.

The following functions are defined using the subtree notation described in the preceding section.

- $F_a(m,k)$ is the minimum total weight of the edges to be removed from the subtree T_a in order to have node a connected to exactly m nodes (including a itself) and k paths surviving in T_a.

- $G_{a_i}(m,k)$ is the minimum total weight of the edges to be removed from the subtree $T_{a_{i,s}} = a + T_{a_i} + T_{a_{i+1}} + \cdots + T_{a_s}$ in order to have a connected to m nodes of $T_{a_{i,s}}$ and k paths surviving in $T_{a_{i,s}}$.

We compute the values for F and G recursively for all $a \in V$, $m = 1, \ldots, n$, $k = 0, \ldots, n(n-1)/2$, as follows. Assume $F_a(m,k) = \infty$, $G_a(m,k) = \infty$ if $m < 0$ or $k < 0$.

$$F_a(m,k) = G_{a_1}(m,k) \qquad \text{for all non-leaf nodes } a \in V, \tag{3.6}$$

$$\begin{aligned} G_{a_i}(m,k) = \min\{&w_e + \min\{F_{a_i}(p,q) + G_{a_{i+1}}(m,k-q) : p = 0, \ldots, \left|V(T_{a_i})\right|,\ q = 0, \ldots, k\}, \\ &\min\{F_{a_i}(p,q) + G_{a_{i+1}}[m-p, k-q-p(m-p)] : p = 0, \ldots, m,\ q = 0, \ldots, k\}\}, \end{aligned} \tag{3.7}$$

where e is the edge connecting a to a_i. Equation (3.7) is written for all non-leaf nodes $a_i \in V$ with $i < s$ (non-rightmost subtrees). For each rightmost subtree T_{a_s} we specify the initial condition

$$G_{a_s}(m,k) = \begin{cases} w_e + \min\{F_{a_s}(p,k) :\ p = 0, \ldots, \left|V(T_{a_s})\right|\} & \text{if } m = 1, \\ F_{a_s}(m-1, k-m+1) & \text{if } m > 1, \end{cases} \tag{3.8}$$

where e is the edge connecting a to a_s, and for every leaf a:

$$F_a(m,k) = \begin{cases} 0 & \text{if } m = 1,\ k = 0, \\ \infty & \text{in all other cases.} \end{cases} \tag{3.9}$$

To explain equations (3.6)–(3.9), we apply the same reasoning as in the preceding section for equations (3.2)–(3.5).

For equation (3.6), note that $T_a = T_{a_{1,s}}$ for any non-leaf node $a \in V$.

For equation (3.7) note that the edge e (which connects a to a_i) is the connecting edge between the subtrees T_{a_i} and $T_{a_{i+1,s}}$. There are two cases to compute the value of $G_{a_i}(m,k)$ based on the edge e, either we remove e or we have to keep it. If the value of $G_{a_i}(m,k)$ is achieved from the expression $w_e + \min\{F_{a_i}(p,q) + G_{a_{i+1}}(m,k-q)\}$, then besides removing the optimal edges from T_{a_i} and $T_{a_{i+1,s}}$ we should also remove the edge e and in this case a path surviving in $T_{a_{i,s}} = T_{a_i} + T_{a_{i+1,s}}$ either completely belongs to T_{a_i} or to $T_{a_{i+1},s}$. According to the definition of $G_{a_i}(m,k)$, m nodes are still connected to a and all nodes are inside the subtree $T_{a_{i+1},s}$, whereas at most $\left|V(T_{a_i})\right|$ nodes can be connected to a_i. If q paths belong to T_{a_i} exactly $k-q$ paths belong to $T_{a_{i+1},s}$. Hence by definition of F and G the minimum total weight of the edges removed from $T_{a_{i,s}}$ will be $G_{a_i}(m,k) = w_e + \min_{p,q}\{F_{a_i}(p,q) + G_{a_{i+1}}(m,k-q)\}$ where w_e corresponds to the weight of the edge e.

On the other hand, if the edge e is not removed, then a is connected to m nodes of T_a and a path in $T_{a_{i,s}}$ can be either completely contained in one of T_{a_i}, $T_{a_{i+1,s}}$, or partially contained in both subtrees because it passes through the edge e. If a_i is connected to p nodes of T_{a_i} and a is connected to $m-p$ nodes in $T_{a_{i+1,s}}$, the paths passing through a are exactly $p(m-p)$. If q paths survive in T_{a_i}, $k-q-p(m-p)$ paths survive in $T_{a_{i+1,s}}$ and, by definition of F and G, $G_{a_i}(m,k) = \min_{p,q}\{F_{a_i}(p,q) + G_{a_{i+1}}[m-p, k-q-p(m-p)]\}$.

The initial condition (3.8) takes into account that, if the edge e (which connects a to a_s) is removed ($m = 1$), the k surviving paths of $T_{a_{s,s}} = a + T_{a_s}$ must belong entirely to T_{a_s}, hence the minimum possible weight for the edges removed from $T_{a_{s,s}}$ will be $G_{a_s}(m,k) = w_e + \min_p\{F_{a_s}(p,k)\}$. On the other hand if the edge e is not removed ($m > 1$), we must have $m-1$ nodes connected to a_s in T_{a_s} and the number of surviving paths is $k-(m-1)$. Thus $G_{a_s}(m,k) = F_{a_s}(m-1, k-m+1)$ follows.

The equation (3.9) says that since $a \in V$ is a leaf, the only possible condition is when $m = 1, k = 0$ and all other combinations of m and k are infeasible and are considered to have an infinite weight.

The optimal value, assuming the tree is rooted at node 1, is given by

$$\mathrm{OPT} = \min\{k\colon F_1(m,k) \le W, m = 1,\dots,n,\ k = 0,\dots,n(n-1)/2\}. \tag{3.10}$$

The optimal solution is recovered by backtracking.

Proposition 3.1.2. *CEDP on a tree with unit connection costs and arbitrary edge weights can be solved by recursion* (3.6)–(3.9) *in* $\mathcal{O}(n^7)$ *time.*

Proof. For each node $a \in V$ there are at most $n+1 = \mathcal{O}(n)$ values for m and $n(n-1)/2+1 = \mathcal{O}(n^2)$ values for k; this gives $\mathcal{O}(n^4)$ values $F_a(\cdot,\cdot)$ and $G_{a_i}(\cdot,\cdot)$ to compute. The heaviest computation lies in equation (3.7), where $\mathcal{O}(n)$ values are possible for p and $\mathcal{O}(n^2)$ for q. Hence in the worst case a number of operations bounded by $\mathcal{O}(n)\cdot\mathcal{O}(n^2)\cdot\mathcal{O}(n^4) = \mathcal{O}(n^7)$ are required.

□

The results derived for the CEDP over trees are summarised in the following table.

c_{uv}	w_v	complexity
$=1$	$=1$	solvable in $\mathcal{O}(n^3K^2)$
$=1$	≥ 0	solvable in $\mathcal{O}(n^7)$

Table 3.1: Complexity results for the CEDP over trees

We will see in Chapter 5 that if the connection costs are not unitary, CEDP on a tree is NP-hard even with unit weights and general 0/1 connection costs.

3.2 Solving CNDP on a tree when given a budget

Di Summa, Grosso, and Locatelli [27] solved the subclass of CNDP over trees in polynomial time when all connections have unit cost with unit node weights and general node weights. We now derive a dynamic programming algorithm for the CNDP over trees considering the fact that we are given a weight limit for removing the nodes and the connection costs are all unitary.

Let us consider the tree $T(V, E)$ with $|V| = n$ nodes, a weight $w_v \geq 0$ for every $v \in V$, a connection cost $c_{uv} = 1$ for all $u, v \in V$ and a weight limit $W \geq 0$. The problem in this case amounts to finding a subset S of nodes with total weight $\sum_{v \in S} w_v$ not exceeding the budget W and minimizing the number of paths surviving in the tree $T(V, E)$ after having removed at most K nodes.

We use the same tree and subtree notation as in Section 3.1, and calculate recursively the following values:

- $F_a(m, k, h)$ = minimum total weight of the k nodes to be removed from the subtree T_a in order to have node a connected to exactly m nodes (including a itself) and h paths surviving in T_a. Condition $m = 0$ indicates that a is removed from T_a and $m > 0$ if a is not removed from T_a.

- $G_{a_i}(m, k, h)$ = minimum total weight of the k nodes to be removed from the subtree $T_{a_{i,s}} = a + T_{a_i} + T_{a_{i+1}} + \cdots + T_{a_s}$ in order to have a connected to m nodes of $T_{a_{i,s}}$ and h paths surviving in $T_{a_{i,s}}$. As above, $m = 0$ indicates that a is removed from $T_{a_{i,s}}$.

We compute the values for F and G recursively for all $a \in V$, $m = 0, \ldots, n$, $h = 0, \ldots, n(n-1)/2$, as follows. We let the function values be infinity whenever the conditions cannot be satisfied. To simplify the recursive formulas below, it will be convenient to accept $h < 0$ in $F_a(m, k, h)$ and $G_{a_i}(m, k, h)$; in this case, we assume again that the function values are infinite.

$$F_a(m, k, h) = G_{a_1}(m, k, h) \qquad \text{for all non-leaf nodes } a \in V, \tag{3.11}$$

$$G_{a_i}(m, k, h) = \begin{cases} \min\{F_{a_i}(p, q, r) + G_{a_{i+1}}(0, k-q, h-r) : \\ \qquad p = 0, \ldots, \left|V(T_{a_i})\right|,\ q = 0, \ldots, k-1,\ r = 0, \ldots, h\} & \text{if } m = 0, \\ \min\{F_{a_i}(p, q, r) + G_{a_{i+1}}[m-p, k-q, h-r-p(m-p)] : \\ \qquad p = 0, \ldots, m-1,\ q = 0, \ldots, k,\ r = 0, \ldots, h\} & \text{if } m > 0. \end{cases} \tag{3.12}$$

Equation (3.12) is written for all non-leaf nodes $a_i \in V$ with $i < s$ (non-rightmost subtrees). For each rightmost subtree T_{a_s} we specify the initial condition

$$G_{a_s}(m, k, h) = \begin{cases} \infty & \text{if } m = 0,\ k = 0, \\ w_a + \min\{F_{a_s}(p, k-1, h) :\ p = 0, \ldots, \left|V(T_{a_s})\right|\} & \text{if } m = 0,\ k > 0, \\ F_{a_s}(m-1, k, h-m+1) & \text{if } m > 0, \end{cases} \tag{3.13}$$

and, for every leaf a:

$$F_a(m,k,h) = \begin{cases} w_a & \text{if } m=0,\ k=1, h=0, \\ 0 & \text{if } m=1,\ k=0, h=0, \\ \infty & \text{in all other cases.} \end{cases} \tag{3.14}$$

We now give a justification for the above formulas. Formula (3.11) is immediate because it follows from the fact that $T_a = T_{a_{1,s}}$ for every non-leaf node $a \in V$.

In the formula for $G_{a_i}(m,k,h)$ (equation (3.12)), the separation is based on whether node a is removed ($m=0$) or not ($m>0$). When $m=0$, a path surviving in $T_{a_{i,s}} = T_{a_i} + T_{a_{i+1,s}}$ either completely belongs to T_{a_i} or to $T_{a_{i+1},s}$. Hence if r paths belong to T_{a_i} exactly $h-r$ belong to $T_{a_{i+1},s}$. In this case if q nodes have been removed from T_{a_i}, exactly $k-q$ nodes (including a) must be removed from $T_{a_{i+1},s}$. Note that the number of nodes p in the subtree T_{a_i} that are connected to a_i is arbitrary, as this value does not affect the number of nodes connected to a, as a is removed in this case and no node will be connected to a in the subtree $T_{a_{i+1},s}$. Hence by definition of F and G the minimum total weight of the nodes removed from $T_{a_{i,s}}$ will be $G_{a_i}(m,k,h) = \min_{p,q,r}\{F_{a_i}(p,q,r) + G_{a_{i+1}}(0,k-q,h-r)\}$. For the other case, if $m>0$ (i.e., a is not removed), a path in $T_{a_{i,s}}$ can be either fully contained in one of T_{a_i}, $T_{a_{i+1,s}}$, or partially contained in both subtrees because it passes through the node a. If a_i is connected to p nodes of T_{a_i} when q nodes have been removed from T_{a_i} and a is connected to $m-p$ nodes in $T_{a_{i+1,s}}$ exactly when $k-q$ nodes to be removed from $T_{a_{i+1,s}}$, the paths passing through a are exactly $p(m-p)$. If r paths survive in T_{a_i}, $h-r-p(m-p)$ paths survive in $T_{a_{i+1,s}}$ and, by definition of F and G, $G_{a_i}(m,k,h) = \min_{p,q,r}\{F_{a_i}(p,q,r)+G_{a_{i+1}}[m-p,k-q,h-r-p(m-p)]\}$.

Formula (3.13) is based on a similar argument. If $m=0$ (i.e., a is removed) and $k>0$, all the h paths survive in T_{a_s} and we have to add the weight of node a. On the other hand if node a is not removed ($m>0$), in T_{a_s} we must have $m-1$ nodes connected to a_s and $h-(m-1)$ surviving paths, since $m-1$ paths connect a to $m-1$ other nodes in $T_{a_{s,s}} = a + T_{a_s}$.

Equation (3.14) reflects the fact that when the subtree consists of just a leaf, the decision to make is whether to remove the leaf ($m=0$, $k=1$) or not ($m=1$, $k=0$), and in both cases the number of paths surviving in T_a is 0.

The optimal value, assuming the tree is rooted at node 1, is given by

$$\text{OPT} = \min\{h\colon F_1(m,K,h) \le W, m=0,\dots,n,\ h=0,\dots,n(n-1)/2\}. \tag{3.15}$$

The optimal solution is recovered by backtracking. We can state the following proposition.

Proposition 3.2.1. *The proposed algorithm has complexity $\mathcal{O}(n^7K^2)$.*

Proof. The number of function values $F_a(\cdot)$ and $G_{a_i}(\cdot)$ to compute for each value of m, k, h is bounded by product $(n+1)(K+1)[n(n-1)/2+1]$ values. The heaviest computation lies in equation (3.12), where $\mathcal{O}(n)$ values are possible for p, $\mathcal{O}(K)$ for q and $\mathcal{O}(n^2)$ for r. Considering all n nodes, the running time is bounded by $\mathcal{O}(n^7K^2)$.

□

3.3 Solving CEDP on a tree when given a budget

In Section 3.1.1 we have solved the CEDP on trees when $c_{uv} = 1$ for all u, v and $w_v = 1$ for all $v \in E$ and in Section 3.1.2 we have also solved the CEDP on trees when $c_{uv} = 1$ for all u, v and $w_v \geq 0$ for all $v \in E$. We now propose a dynamic programming technique for the CEDP over trees, taking into account the weight limit for removing edges and the fact that all connection costs are unitary.

Let us consider the tree $T(V, E)$ with $|V| = n$ nodes, a weight $w_v \geq 0$ for every $v \in E$, a connection cost $c_{uv} = 1$ for all $u, v \in V$ and a weight limit $W \geq 0$. The problem in this case amounts to finding a subset S of edges with total weight $\sum_{v \in S} w_v$ not exceeding the budget W and minimizing the number of paths surviving in a tree $T(V, E)$ after having removed at most K edges.

As in Section 3.1, we utilize the same tree and subtree notation. In order to solve the problem by dynamic programming, we define the following functions.

- $F_a(m, k, h)$ = minimum total weight of the k edges to be removed from the subtree T_a in order to have node a connected to exactly m nodes (including a itself) and h paths surviving in T_a. Note that the number of nodes connected to the root will never be 0 because we never remove the root as the root is a node and we can not remove a node in the edge deletion problem.

- $G_{a_i}(m, k, h)$ = minimum total weight of the k edges to be removed from the subtree $T_{a_{i,s}} = a + T_{a_i} + T_{a_{i+1}} + \cdots + T_{a_s}$ in order to have a connected to m nodes of $T_{a_{i,s}}$ and h paths surviving in $T_{a_{i,s}}$. As above, $m > 0$.

We compute the values for F and G recursively for all $a \in V$, $m = 1, \ldots, n$, $h = 0, \ldots, n(n-$

1)/2, as follows. Assume $F_a(m,k,h) = \infty$, $G_a(m,k,h) = \infty$ if $m < 0$ or $h < 0$.

$$F_a(m,k,h) = G_{a_1}(m,k,h) \qquad \text{for all non-leaf nodes } a \in V, \tag{3.16}$$

$$G_{a_i}(m,k,h) = \begin{cases} \min\{F_{a_i}(p,0,r) + G_{a_{i+1}}[m-p, 0, h-r-p(m-p)] : \\ \qquad p = 0,\dots,m,\ r = 0,\dots,h\} & \text{if } k = 0, \\ \min\{w_e + \min\{F_{a_i}(p,q,r) + G_{a_{i+1}}(m, k-q-1, h-r) : \\ \qquad p = 0,\dots,\left|V(T_{a_i})\right|,\ q = 0,\dots,k-1,\ r = 0,\dots,h\}, \\ \qquad \min\{F_{a_i}(p,q,r) + G_{a_{i+1}}[m-p, k-q, h-r-p(m-p)] : \\ \qquad p = 0,\dots,m,\ q = 0,\dots,k,\ r = 0,\dots,h\}\} & \text{if } k > 0, \end{cases} \tag{3.17}$$

where e is the edge connecting a to a_i. Equation (3.17) is written for all non-leaf nodes $a_i \in V$ with $i < s$ (non-rightmost subtrees). For each rightmost subtree T_{a_s} we specify the initial condition

$$G_{a_s}(m,k,h) = \begin{cases} \infty & \text{if } m = 1,\ k = 0, \\ w_e + \min\{F_{a_s}(p, k-1, h) : p = 0,\dots,\left|V(T_{a_s})\right|\} & \text{if } m = 1,\ k > 0, \\ F_{a_s}(m-1, k, h-m+1) & \text{if } m > 1,\ k \geq 0, \end{cases} \tag{3.18}$$

where e is the edge connecting a to a_s, and for every leaf a:

$$F_a(m,k,h) = \begin{cases} 0 & \text{if } m = 1,\ k = 0, h = 0, \\ \infty & \text{in all other cases.} \end{cases} \tag{3.19}$$

We will now explain why the formulas above are correct. Formula (3.16) is immediate because it derives from the fact that $T_a = T_{a_{1,s}}$ for any non-leaf node $a \in V$.

For equation (3.17) note that the edge e (which connects a to a_i) is the connecting edge between the subtrees T_{a_i} and $T_{a_{i+1,s}}$. The first case $k = 0$ means that we are not allowed to remove any edge from $T_{a_{i,s}} = T_{a_i} + T_{a_{i+1,s}}$, hence if a_i is connected to p nodes of T_{a_i} and a is connected to $m-p$ nodes in $T_{a_{i+1,s}}$, the paths passing through the edge e are exactly $p(m-p)$. If r paths survive in T_{a_i}, $h - r - p(m-p)$ paths survive in $T_{a_{i+1,s}}$ and, by definition of F and G, $G_{a_i}(m,0,h) = \min_{p,r}\{F_{a_i}(p,0,r) + G_{a_{i+1}}[m-p, 0, h-r-p(m-p)]\}$. For the second case in which $k > 0$, we take the better of two possibilities, which correspond to the two arguments of the outer minimum. The first possibility occurs only when we remove the connecting edge a to a_i and in this situation we take the sum of the optimal values that we can obtain in each of the two subtrees T_{a_i} and $T_{a_{i+1,s}}$. For the second possibility, which is when we are not removing the edge e, a path is either fully contained in one of T_{a_i}, $T_{a_{i+1,s}}$, or partially contained in both subtrees because it passes through the edge e. If a_i is connected to p nodes of T_{a_i} when q edges have been removed from T_{a_i} and a is connected to $m-p$ nodes in $T_{a_{i+1,s}}$ exactly when

$k-q$ edges have been removed from $T_{a_{i+1},s}$, the paths passing through the edge e are exactly $p(m-p)$. If r paths survive in T_{a_i}, then $h-r-p(m-p)$ paths survive in $T_{a_{i+1},s}$ and, by definition of F and G, $G_{a_i}(m,k,h) = \min_{p,q,r}\{F_{a_i}(p,q,r)+G_{a_{i+1}}[m-p,k-q,h-r-p(m-p)]\}$.

The initial condition (3.18) takes into account that, if the edge e is removed ($m=1$) and $k>0$, the h surviving paths of $T_{a_s,s} = a+T_{a_s}$ must belong entirely to T_{a_s}, hence the minimum possible weight for the edges removed from $T_{a_s,s}$ will be $G_{a_s}(m,k,h) = w_e + \min_p\{F_{a_s}(p,k-1,h)\}$. On the other hand if the edge e is not removed ($m>1$), in T_{a_s} we must have $m-1$ nodes connected to a_s and $h-(m-1)$ surviving paths, since $m-1$ paths connect a to $m-1$ other nodes in $T_{a_s,s}$; thus $G_{a_s}(m,k,h) = F_{a_s}(m-1,k,h-m+1)$ follows.

Equation (3.19) handles the case of a one-node tree. Since $a \in V$ is a leaf, it is not possible to remove any edge ($k=0$) and only a is connected to itself ($m=1$) and the number of paths h surviving in T_a is 0.

The optimal value, assuming the tree is rooted at node 1, is given by

$$\mathrm{OPT} = \min\{h\colon F_1(m,K,h) \le W, m=0,\dots,n,\ h=0,\dots,n(n-1)/2\}. \tag{3.20}$$

The optimal solution is recovered by backtracking. We can make the following proposition.

Proposition 3.3.1. *The proposed algorithm has complexity $\mathcal{O}(n^7K^2)$.*

Proof. The number of function values $F_a(\cdot)$ and $G_{a_i}(\cdot)$ to compute for each value of m,k,h is bounded by product $(n+1)(K+1)[n(n-1)/2+1]$ values. The heaviest computation lies in equation (3.17), where $\mathcal{O}(n)$ values are possible for p, $\mathcal{O}(K)$ for q and $\mathcal{O}(n^2)$ for r. Considering all n nodes, the running time is bounded by $\mathcal{O}(n^7K^2)$.

□

3.4 Solving CNEDP-1 on a tree with unit connection costs

In this section we show CNEDP-1 can be solved in polynomial time when the underlying graph is a tree and the connection costs are all unitary and node and edge weights are arbitrary. We would like to mention that the case with unit costs and unit node and edge weights does not make sense because it is always better to use the budget to remove nodes rather than edges.

Let $w_v \ge 0$ and $w_e \ge 0$ be arbitrary weights assigned to the nodes $v \in V$ and edges $e \in E$ respectively. The problem in this case amounts to finding subset $S \subseteq V \cup E$ with total weight $\sum_{v\in V} w_v + \sum_{e\in E} w_e$ not exceeding a given W such that the number of surviving paths after having removed nodes and edges is minimized.

This case can be solved by a dynamic programming algorithm formulated in the same spirit of Section 3.1. The recursion uses two parameters m and k representing, respectively,

the number of nodes connected to the root of a subtree and the number of paths surviving in the same subtree.

Keeping the notation for subtrees introduced in Section 3.1, we define the following functions.

- $F_a(m,k)$ is the minimum total weight of the nodes and edges to be removed from the subtree T_a in order to have node a connected to exactly m nodes (including a itself) and k paths surviving in T_a.
- $G_{a_i}(m,k)$ is the minimum total weight of the nodes and edges to be removed from the subtree $T_{a_{i,s}} = a + T_{a_i} + T_{a_{i+1}} + \cdots + T_{a_s}$ in order to have a connected to m nodes of $T_{a_{i,s}}$ and k paths surviving in $T_{a_{i,s}}$.

We compute the values for F and G recursively for all $a \in V$, $m = 0, \ldots, n$, $k = 0, \ldots, n(n-1)/2$, as follows. For both functions, condition $m = 0$ implies that node a is deleted from the graph ($a \in S$). Also, in case no feasible solution exists for functions F_a or G_{a_i}, we will set the corresponding entries to ∞. It is also convenient to set the above values to infinity when $k < 0$.

We can state the following recursive relations:

$$F_a(m,k) = G_{a_1}(m,k) \qquad \text{for all non-leaf nodes } a \in V, \tag{3.21}$$

$$G_{a_i}(m,k) = \begin{cases} \min\{F_{a_i}(p,q) + G_{a_{i+1}}(0,k-q) : \\ \qquad p = 0, \ldots, \left|V(T_{a_i})\right|,\ q = 0, \ldots, k\} & \text{if } m = 0, \\ \min\{w_e + \min\{F_{a_i}(p,q) + G_{a_{i+1}}(m,k-q) : \\ \qquad p = 0, \ldots, \left|V(T_{a_i})\right|,\ q = 0, \ldots, k\}, \\ \qquad \min\{F_{a_i}(p,q) + G_{a_{i+1}}[m-p, k-q-p(m-p)] : \\ \qquad p = 0, \ldots, m,\ q = 0, \ldots, k\}\} & \text{if } m > 0, \end{cases} \tag{3.22}$$

where e is the edge connecting a to a_i. Equation (3.22) is written for all non-leaf nodes $a_i \in V$ with $i < s$ (non-rightmost subtrees). For each rightmost subtree T_{a_s} we specify the initial condition

$$G_{a_s}(m,k) = \begin{cases} w_a + \min\{F_{a_s}(p,k) :\ p = 0, \ldots, \left|V(T_{a_s})\right|\} & \text{if } m = 0, \\ w_e + \min\{F_{a_s}(p,k) :\ p = 0, \ldots, \left|V(T_{a_s})\right|\} & \text{if } m = 1, \\ F_{a_s}(m-1, k-m+1) & \text{if } m > 1, \end{cases} \tag{3.23}$$

where e is the edge connecting a to a_s, and for every leaf a:

$$F_a(m,k) = \begin{cases} w_a & \text{if } m = 0,\ k = 0, \\ 0 & \text{if } m = 1,\ k = 0, \\ \infty & \text{in all other cases.} \end{cases} \tag{3.24}$$

We now give a justification for the above formulas.

For equation (3.21), note that $T_a = T_{a_{1,s}}$ for any non-leaf node $a \in V$.

For equation (3.22), if $m = 0$ (i.e. a is removed), a path surviving in $T_{a_{i,s}} = T_{a_i} + T_{a_{i+1,s}}$ either belongs entirely to T_{a_i} or to $T_{a_{i+1},s}$, hence if q paths belong to T_{a_i} exactly $k - q$ paths belong to $T_{a_{i+1},s}$ and p nodes of T_{a_i} can be connected to a_i, whereas no node will be connected to a. Hence by definition of F and G the minimum total weight of the nodes and edges removed from $T_{a_{i,s}}$ will be $G_{a_i}(m,k) = \min_{p,q}\{F_{a_i}(p,q) + G_{a_{i+1}}(0,k-q)\}$.

Now consider the case when a is not removed ($m > 0$). Note that the edge e (which connects a to a_i) is the connecting edge between the subtrees T_{a_i} and $T_{a_{i+1,s}}$. There are two cases to compute the value of $G_{a_i}(m,k)$ based on the edge e, either we remove e or we have to keep it. If the value of $G_{a_i}(m,k)$ is achieved from the expression $w_{a,a_i} + \min\{F_{a_i}(p,q) + G_{a_{i+1}}(m,k-q)\}$, then besides removing the optimal nodes and edges from T_{a_i} and $T_{a_{i+1,s}}$ we should also remove the edge e and in this case a path surviving in $T_{a_{i,s}} = T_{a_i} + T_{a_{i+1,s}}$ either completely belongs to T_{a_i} or to $T_{a_{i+1},s}$. According to the definition of $G_{a_i}(m,k)$, m nodes are still connected to a and all nodes are inside the subtree $T_{a_{i+1},s}$, whereas at most $\left|V(T_{a_i})\right|$ nodes can be connected to a_i. If q paths belong to T_{a_i} exactly $k - q$ paths belong to $T_{a_{i+1},s}$. Hence by definition of F and G the minimum total weight of the nodes and edges removed from $T_{a_{i,s}}$ will be $G_{a_i}(m,k) = w_e + \min_{p,q}\{F_{a_i}(p,q) + G_{a_{i+1}}(m,k-q)\}$ where w_e corresponds the weight of the edge e.

On the other hand if the edge e is not removed, then a is connected to m nodes of T_a and a path in $T_{a_{i,s}}$ can be either completely contained in one of T_{a_i}, $T_{a_{i+1,s}}$, or partially contained in both subtrees because it passes through a. If a_i is connected to p nodes of T_{a_i} and a is connected to $m - p$ nodes in $T_{a_{i+1,s}}$, the paths passing through a are exactly $p(m-p)$. If q paths survive in T_{a_i}, $k - q - p(m-p)$ paths survive in $T_{a_{i+1,s}}$ and, by definition of F and G, $G_{a_i}(m,k) = \min_{p,q}\{F_{a_i}(p,q) + G_{a_{i+1}}[m-p, k-q-p(m-p)]\}$.

The initial condition (3.23) takes into account that, if node a is removed ($m = 0$), the k surviving paths of $T_{a_{s,s}} = a + T_{a_s}$ must belong entirely to T_{a_s}, hence the minimum possible weight for the nodes and edges removed from $T_{a_{s,s}}$ will be $G_{a_s}(0,k) = \min_p\{F_{a_s}(p,k)\} + w_a$. When $m = 1$, i.e., the edge e (which connects a to a_s) is removed, then the k surviving paths of $T_{a_{s,s}} = a + T_{a_s}$ must belong entirely to T_{a_s}, and hence the minimum possible weight for the nodes and edges removed from $T_{a_{s,s}}$ will be $G_{a_s}(m,k) = w_e + \min_p\{F_{a_s}(p,k)\}$. On the other hand if the edge e is not removed ($m > 1$), in T_{a_s} we must have $m - 1$ nodes connected to a_s and $k - (m-1)$ surviving paths, since $m - 1$ paths connect a to $m - 1$ other nodes in $T_{a_{s,s}}$; thus $G_{a_s}(m,k) = F_{a_s}(m-1, k-m+1)$ follows.

The equation (3.24) trivially handles the case of a one-node tree: remove the single node a (case $m = 0, k = 0$), or keep it ($m = 1, k = 0$) and all other combinations of m and k are infeasible and are considered to have an infinite weight.

The optimal value, assuming the tree is rooted at node 1, is given by

$$\mathrm{OPT} = \min\{k\colon F_1(m,k) \le W, m = 0,\dots,n,\ k = 0,\dots,n(n-1)/2\}. \tag{3.25}$$

The optimal solution is recovered by backtracking.

Proposition 3.4.1. *CNEDP-1 on a tree with unit connection costs and arbitrary node and edge weights can be solved by recursion* (3.21)–(3.24) *in* $\mathcal{O}(n^7)$ *time.*

Proof. For each node $a \in V$ there are at most $n+1 = \mathcal{O}(n)$ values for m and $n(n-1)/2+1 = \mathcal{O}(n^2)$ values for k; this gives $\mathcal{O}(n^4)$ values $F_a(\cdot,\cdot)$ and $G_{a_i}(\cdot,\cdot)$ to compute. The heaviest computation lies in equation (3.22), where $\mathcal{O}(n)$ values are possible for p and $\mathcal{O}(n^2)$ for q. Hence in the worst case a number of operations bounded by $\mathcal{O}(n^7)$ are required.

□

3.5 Complexity of CNEDP-2

Unlike CNEDP-1, CNEDP-2 with unit connection costs is NP-hard even on a path. We actually show a stronger result: it is NP-hard to approximate this problem within any factor. (We recall that given $\alpha \ge 1$, an α-approximation algorithm for a minimization problem is required to return a solution whose objective value is at most α times the optimal value; see, e.g., [76].)

Proposition 3.5.1. *Unless* $P = NP$, CNEDP-2 *on a path with unit connection costs cannot be approximated within any factor.*

Proof. We prove the result via a reduction from Partition, which is known to be NP-complete (see [36]):

> Partition: Given $n \in \mathbb{N}$ and $a_1,\dots,a_n \in \mathbb{N}$, determine whether there exists $J \subseteq \{1,\dots,n\}$ such that $\sum_{i\in J} a_i = \sum_{i\notin J} a_i$.

Given an instance of Partition as above, we define $A = \sum_{i=1}^{n} a_i$. We construct an instance of CNEDP-2 on a path as follows:

- $G = (V,E)$ is a path with $2n+1$ vertices, denoted by $u_1, v_1, u_2, v_2, \dots, u_n, v_n, u_{n+1}$ (in the order they appear on the path);
- the node weights are $w_{v_i} = a_i$ for every $i \in \{1,\dots,n\}$, and $w_{u_i} = A/2+1$ for every $i \in \{1,\dots,n+1\}$;
- the edge weights are $w_{u_iv_i} = a_i$ and $w_{v_iu_{i+1}} = 0$ for every $i \in \{1,\dots,n\}$;
- the weight limits are $W_V = W_E = A/2$.

We show that the given instance of PARTITION has a solution if and only if the optimal value of the above instance of CNEDP-2 is zero. Since, by definition, an approximation algorithm always recognizes the instances with optimal value equal to zero, this will prove the result.

Assume that the instance of PARTITION has a solution, i.e., there exists $J \subseteq \{1, \ldots, n\}$ such that $\sum_{i \in J} a_i = \sum_{i \notin J} a_i = A/2$. Define $S \subseteq V \cup E$ as follows:

$$S = \{v_i : i \in J\} \cup \{u_i v_i : i \notin J\} \cup \{v_i u_{i+1} : i \in \{1, \ldots, n\}\}.$$

This choice yields a feasible solution to the instance of CNEDP-2, as $w(S \cap V) = w(S \cap E) = A/2 = W_V = W_E$. Furthermore, as $G - S$ contains only isolated nodes, $c(G - S) = 0$. Thus the optimal value of the instance of CNEDP-2 is zero.

For the converse, assume that the instance of CNEDP-2 has a solution with objective value zero, i.e., there exists $S \subseteq V \cup E$ such that $w(S \cap V) \leq A/2$, $w(S \cap E) \leq A/2$, and $c(G - S) = 0$ (which means that there are only isolated nodes in $G - S$). Without loss of generality, we can assume that $v_i u_{i+1} \in S$ for every $i \in \{1, \ldots, n\}$, as these edges have zero weight. In other words, we can imagine that the original graph only contains the edges $u_1 v_1, \ldots, u_n v_n$. Furthermore, $u_i \notin S$ for every i, as otherwise the node weight limit would be exceeded. Then the fact that $G - S$ has only isolated nodes implies that, for every $i \in \{1, \ldots, n\}$, S contains v_i or $u_i v_i$. As $w_{v_i} = w_{u_i v_i} = a_i$ for every i, we obtain $w(S) \geq A$. Since the weight limits are $W_V = W_E = A/2$, we necessarily have $w(S \cap V) = w(S \cap E) = A/2$. Thus the instance of PARTITION has the solution $J = S \cap V$. □

3.5.1 Solving CNEDP-2 on a tree with unit connection costs

Despite the above negative result, we now show that if the node and edge weights are fixed to 1 (as well as the connection costs), CNEDP-2 admits a polynomial-time algorithm. The objective in this case is for minimizing the number of paths surviving in the tree $T(V, E)$ after having removed at most K_V nodes and K_E edges.

We use the same tree and subtree notation as in Section 3.1, and calculate recursively the following functions:

- $F_a(m, k_V, k_E)$ = minimum number of connections that still exists in the subtree T_a after k_V nodes and k_E edges have been removed from T_a and m nodes of T_a remains connected to the root a. Condition $m = 0$ indicates that a is removed from T_a.

- $G_{a_i}(m, k_V, k_E)$ minimum number of connections that still exists in the subtree $T_{a_{i,s}} = a + T_{a_i} + \cdots + T_{a_s}$ after k_V nodes and k_E edges have been removed from $T_{a_{i,s}}$ and m nodes of the subtree are still connected to a. As above, $m = 0$ indicates that a is removed from $T_{a_{i,s}}$.

We let the function values be infinity whenever the conditions cannot be satisfied. The values for F and G can be computed by traversing the tree in postorder (from leaves to root), by means of the following relations:

For every leaf a we have

$$F_a(m, k_V, k_E) = \begin{cases} 0 & \text{if } (m = 1,\ k_V = k_E = 0) \text{ or } (m = k_E = 0,\ k_V = 1), \\ \infty & \text{otherwise,} \end{cases} \tag{3.26}$$

while the formula for a non-leaf node a is

$$F_a(m, k_V, k_E) = G_{a_1}(m, k_V, k_E). \tag{3.27}$$

If a is a non-leaf node, we also have

$$G_{a_s}(m, k_V, k_E) = \begin{cases} \min\{F_{a_s}(p, k_V - 1, k_E) : 0 \le p \le |V(T_{a_s})|\} & \text{if } m = 0, \\ \min\left\{F_{a_s}(0, k_V, k_E), \min\{F_{a_s}(p, k_V, k_E - 1) : 0 \le p \le |V(T_{a_s})|\}\right\} & \text{if } m = 1, \\ F_{a_s}(m - 1, k_V, k_E) + m - 1 & \text{if } m > 1, \end{cases} \tag{3.28}$$

while, for $i < s$,

$$G_{a_i}(m, k_V, k_E) = \begin{cases} \min\{F_{a_i}(p, q_V, q_E) + G_{a_{i+1}}(0, k_V - q_V, k_E - q_E) : \\ \qquad 0 \le p \le |V(T_{a_i})|,\ 0 \le q_V \le k_V - 1,\ 0 \le q_E \le k_E\} & \text{if } m = 0, \\ \min\Big\{ \min\{F_{a_i}(p, q_V, q_E) + G_{a_{i+1}}(m, k_V - q_V, k_E - q_E - 1) : \\ \qquad\qquad 0 \le p \le |V(T_{a_i})|,\ 0 \le q_V \le k_V,\ 0 \le q_E \le k_E - 1\}, \\ \qquad \min\{F_{a_i}(p, q_V, q_E) + G_{a_{i+1}}(m - p, k_V - q_V, k_E - q_E) + p(m - p) : \\ \qquad\qquad 0 \le p \le m,\ 0 \le q_V \le k_V,\ 0 \le q_E \le k_E\}\Big\} & \text{if } m > 0. \end{cases} \tag{3.29}$$

The optimal value is calculated as follows if we denote by 1 the root node of the tree

$$\mathrm{OPT} = \min\{F_1(m, K_V, K_E) : m = 0, \ldots, n\}. \tag{3.30}$$

Formulas (3.26) and (3.27) are immediate.

In (3.28) we assume that if $a \in S$ then $aa_s \notin S$: This is without loss of generality, as if $aa_s \in S$, we obtain the same objective value by removing the elements in $S \setminus \{aa_s\}$. The case $m = 0$ corresponds to $a \in S$, which leads to the formula on the first line. The case $m = 1$ occurs when $a_s \in S$ (first argument of the outer minimum on the second line) or $aa_s \in S$ (second argument of the outer minimum). Finally, $m > 1$ is the case in which $a, aa_s \notin S$.

Similar to (3.28), in (3.29) we assume that if $a \in S$ then $aa_i \notin S$. The first case ($m = 0$) corresponds to having $a \in S$. In this situation, we take the sum of the optimal values that we can obtain in each of the two subtrees T_{a_i} and $T_{a_{i+1,s}}$. For the second case ($m > 0$), in which $a \notin S$, we take the better of two possibilities, which correspond to the two arguments of the outer minimum. For the first possibility, which is when $aa_i \in S$, we take again the sum of the optimal values in each of the two subtrees T_{a_i} and $T_{a_{i+1,s}}$. For the second possibility ($aa_i \notin S$), we have to add the connections between the two subtrees.

We obtain the following result.

Proposition 3.5.2. CNEDP-2 *on a tree with unit connection costs and unit node/edge weights can be solved in* $\mathcal{O}(n^3K_V^2K_E^2)$ *time.*

Proof. For each node $a \in V$ there are at most $n + 1 = \mathcal{O}(n)$ values for m, $K_V + 1 = \mathcal{O}(K_V)$ values for k_V and $K_E + 1 = \mathcal{O}(K_E)$ values for k_E; this gives $\mathcal{O}(n^2K_VK_E)$ values of F and G to compute. The heaviest computation is that of equation (3.29) that requires at most $\mathcal{O}(nK_VK_E)$ steps. Hence in the worst case a number of operations bounded by $\mathcal{O}(n^3K_V^2K_E^2)$ are required. □

The results derived for the CNEDP-2 over trees are summarised in the following table.

c_{uv}	w_v	w_e	complexity
$=1$	$=1$	$=1$	$\mathcal{O}(n^3K_V^2K_E^2)$
$=1$	≥ 0	≥ 0	$\mathcal{NP}$-hard

Table 3.2: Complexity results for the CNEDP-2 over trees

Chapter 4

CEDP via subdivision

In Chapter 3 we have derived the dynamic programming algorithm to solve the CEDP on a tree by removing the edges directly. In this chapter we propose a different approach to solve the CEDP over trees by looking at the CNDP by subdividing the graph and the same complexity (see Section 3.1.1 and 3.1.2) is obtained. In a graph $G = (V, E)$, the subdivision of an edge $e = uv \in E(G)$ means the substitution of the edge e by a vertex w and the new edges uw and wv (see Figure 4.1). This operation generates a new graph G':

$$G' = (V \cup \{w\}, (E \setminus \{uv\}) \cup \{uw, wv\}).$$

Figure 4.1: Example of subdivision of an edge uv.

The technique that we follow in this chapter to solve the CEDP over trees is the following: Let $G = (V, E)$ be an undirected graph with a finite set V of nodes and a finite set $E \subseteq V \times V$ of edges. Define a restricted critical node detection problem (RCNDP) in which we are also given a subset of vertices $A \subseteq V$ and the problem is to disconnect the graph as much as possible by removing K nodes from set A. Now we create the graph $G' = (V', E')$ in which all the edges are subdivided and define A to be the set of newly created nodes where $V' = V \cup A$. Then the problem of removing at most K edges from G is equivalent to a RCNDP on G', i.e., the problem of removing at most K nodes from the set $A \subseteq V'$.

In the above example (Figure 4.1), removing the edge uv is the same as removing the newly created node w in the subdivided graph because when we remove the node w, this automatically remove the two new edges uw and wv.

4.1 The unit cost, unit weight case on trees

An approach for solving CEDP on trees when $c_{uv} = 1$ for all u, v and $w_v = 1$ for all $v \in V$ is shown in this section.

Let us consider the tree $T(V, E)$ with $|V| = n$ nodes. We have created the tree $T' = (V', E')$ by subdividing every edge of T. Now we consider the RCNDP on T' with the allowed set $A = V' \setminus V$, i.e., the set of newly created vertices. In this case the problem calls for minimizing the number of paths surviving in the tree $T'(V', E')$ after having removed at most K nodes from the set A and we want to count the connections between the original vertices, i.e., $c_{uv} = 1$ for all $u, v \in V$ and 0 otherwise.

We use the same tree and subtree notation as in Section 3.1, and calculate recursively the following values:

- $F_a(m, k)$ = minimum number of connections between pairs of nodes in V that still exists in the subtree T_a after k nodes from A have been removed from T_a and m original nodes of T_a remains connected to the root a (including a itself).

- $G_{a_i}(m, k)$ = minimum number of connections between pairs of nodes in V that still exists in the subtree $T_{a_{i,s}} = a + T_{a_i} + \cdots + T_{a_s}$ after k nodes from A have been removed from $T_{a_{i,s}}$ and m original nodes of the subtree are still connected to a.

We remark that for both functions, the number of the original nodes connected to the root will never be 0 (i.e. $m > 0$) because we never remove the root as the root is an original node and we can not remove an original node in the restricted problem. Furthermore, whenever the conditions in one of the above definitions cannot be satisfied, we set the value of the function to infinity.

The values for F and G can be computed by traversing the tree in postorder (from leaves to root), by means of the following relations:

$$F_a(m, k) = G_{a_1}(m, k) \quad \text{for any non-leaf node } a \in V; \tag{4.1}$$

$$G_{a_i}(m, k) = \begin{cases} \min\{F_{a_i}(p, 0) + G_{a_{i+1}}(m - p, 0) + p(m - p) : p = 0, \ldots, m\} & \text{if } k = 0, \\ \min\{\min\{F_{a_i}(p, q) + G_{a_{i+1}}(m, k - 1 - q) : p = 0, \ldots, \left|V(T_{a_i})\right|,\ q = 0, \ldots, k - 1\}, \\ \qquad \min\{F_{a_i}(p, q) + G_{a_{i+1}}(m - p, k - q) + p(m - p) : \\ \qquad p = 0, \ldots, m,\ q = 0, \ldots, k\}\} & \text{if } k > 0, \end{cases} \tag{4.2}$$

for any non-leaf node $a \in V$ and $i < s$.

The initial conditions on each leaf a and on each rightmost subtree T_{a_s} are the following:

$$F_a(m,k) = \begin{cases} 0 & \text{if } m = 1, k = 0, \\ \infty & \text{otherwise,} \end{cases} \tag{4.3}$$

$$G_{a_s}(m,k) = \begin{cases} \infty & \text{if } m = 1,\ k = 0, \\ \min\{F_{a_s}(p,k-1) : p = 0,\ldots,|V(T_{a_s})|\} & \text{if } m = 1,\ k > 0, \\ F_{a_s}(m-1,k) + (m-1) & \text{if } m > 1,\ k \geq 0, \end{cases} \tag{4.4}$$

Equation (4.1) follows because $T_a = T_{a_{1,s}}$ for any non-leaf node $a \in V$.

Recursion (4.2) can be interpreted as follows:

The case $k = 0$ means that we are not removing any node from $T_{a_{i,s}} = T_{a_i} + T_{a_{i+1,s}}$. Since we have to keep everything, we are not allowed to remove anything from the subtrees T_{a_i} and $T_{a_{i+1,s}}$. If a_i is connected to p original nodes of T_{a_i}, then a is connected to $m-p$ original nodes in $T_{a_{i+1,s}}$ and the paths passing through a are exactly $p(m-p)$. Hence by definition of F and G the minimum number of paths that survive in $T_{a_{i,s}}$ when we are not removing anything will be $G_{a_i}(m,0) = \min\{F_{a_i}(p,0) + G_{a_{i+1}}(m-p,0) + p(m-p)\}$ where $p = 0,\ldots,m$.

The case $k > 0$ means that we have to remove at least one node from the allowed set A. When the value of $G_{a_i}(m,k)$ is achieved from the expression $F_{a_i}(p,q) + G_{a_{i+1}}(m,k-1-q)$, we remove the node corresponding to the edge e which connects a to a_i. Expression $F_{a_i}(p,q)$ gives the minimum number of paths that survive in T_{a_i} when q nodes from A are removed from T_{a_i} and p original nodes of T_{a_i} are still connected to a_i. Since q nodes from A have been removed from T_{a_i}, exactly $k-1-q$ nodes from A must be removed from $T_{a_{i+1,s}}$ and m original nodes of $T_{a_{i+1,s}}$ are still connected to a. The minimum number of paths that survive in $T_{a_{i+1,s}}$ when $k-1-q$ nodes from A are removed from $T_{a_{i+1,s}}$ is given by $G_{a_{i+1}}(m,k-1-q)$. Thus the expression $F_{a_i}(p,q) + G_{a_{i+1}}(m,k-1-q)$ gives the minimum number of paths that survive in $T_{a_{i,s}}$ when q nodes from A are removed from T_{a_i} (and the other $k-1-q$ nodes from A are removed from $T_{a_{i+1,s}}$) and p original nodes of T_{a_i} are still connected to a_i. By taking the minimum over $p = 0,\ldots,|V(T_{a_i})|$ and $q = 0,\ldots,k-1$, we find the value of $G_{a_i}(m,k)$.

When the value of $G_{a_i}(m,k)$ is achieved from the expression $F_{a_i}(p,q) + G_{a_{i+1}}(m-p,k-q) + p(m-p)$, we are not removing the node corresponding to the edge e. As above, expression $F_{a_i}(p,q)$ gives the minimum number of paths that survive in T_{a_i} when q nodes from A are removed from T_{a_i} and p original nodes of T_{a_i} are still connected to a_i. Since q nodes from A have been removed from T_{a_i}, exactly $k-q$ nodes from A must be removed from $T_{a_{i+1,s}}$ and since p original nodes of T_{a_i} are still connected to a_i and thus to a, exactly $m-p$ original nodes of $T_{a_{i+1,s}}$ must remain connected to a. The minimum number of paths that survive in $T_{a_{i+1,s}}$ when $k-q$ nodes from A are removed from $T_{a_{i+1,s}}$ and $m-p$ original nodes of $T_{a_{i+1,s}}$ are still connected to a is given by $G_{a_{i+1}}(m-p,k-q)$. Thus the expression $F_{a_i}(p,q) + G_{a_{i+1}}(m-p,k-q)$ gives the minimum number of paths that survive in T_{a_i} or $T_{a_{i+1,s}}$ when q nodes from A are

removed from T_{a_i} (and the other $k-q$ nodes from A are removed from $T_{a_{i+1,s}}$) and p original nodes of T_{a_i} are still connected to a_i, while $m-p$ original nodes of $T_{a_{i+1,s}}$ are still connected to a. Now we have to add the paths connecting to the original nodes of T_{a_i} to the original nodes of $T_{a_{i+1,s}}$, i.e. $p(m-p)$ paths. This gives expression $F_{a_i}(p,q)+G_{a_{i+1}}(m-p,k-q)+p(m-p)$ of recursion (4.2). By taking the minimum over $p=0,\ldots,m$ and $q=0,\ldots,k$, we find the value of $G_{a_i}(m,k)$.

Equation (4.3) handles the case of a one-node tree. Since $a \in V$ is a leaf and it is not in the allowed set A, it is not possible to remove the node ($k=0$) and only a is connected to itself ($m=1$) and the number of paths survive in T_a is 0.

For a justification of (4.4), recall that $T_{a_{s,s}} = a + T_{a_s}$. If $m=1$ and $k>0$, then we have to remove the node corresponding to the edge between a and a_s and the other $k-1$ nodes we have to remove from the subtree T_{a_s} from A and the number of connections that survive are those in T_{a_s} between the original vertices. On the other hand if $m>1$, then we can not remove the node corresponding to the edge a to a_s and in this time we have to remove all the k nodes from A inside the subtree T_{a_s}. Since m original nodes connected to a including a itself, we will find the other $m-1$ original nodes connected to a_s. Then we have to add all the connections of a to the original nodes that are connected to a_s in the subtree.

The optimal value for the problem, assuming that the tree T is rooted at node 1, is given by

$$\mathrm{OPT} = \min\{F_1(m,K) : m = 1,\ldots,n\}, \tag{4.5}$$

and the optimal solution is recovered by backtracking.

Proposition 4.1.1. *CEDP on a tree with unit connection costs and unit edge weights can be solved by recursion* (4.1)–(4.4) *in* $\mathcal{O}(n^3K^2)$ *time.*

4.2 The case with unit costs and arbitrary edge weights

In this section, we look at the identical problem as in Section 4.1, but with arbitrary edge weights rather than unit edge weights.

Let us consider the tree $T(V,E)$ with $|V|=n$ nodes. We have created the tree $T'=(V',E')$ in which all the edges are subdivided. Now we consider the restricted CNDP on T' with the allowed set $A = V' \setminus V$, i.e., the set of newly created vertices. Once the subdivision has been made, all the vertices keep the same cost as the original vertices and the new vertices inherit the cost of the edges from which they are subdividing because every vertex subdivides an edge. Let $w_v \geq 0$ be arbitrary weights assigned to the newly created nodes $v \in A$. The problem in this case amounts to finding a subset S of nodes from the set A with total weight $\sum_{v\in S} w_v$ not exceeding a given W such that the number of surviving paths after having removed the

set S is minimized and we want to count the connections between the original vertices, i.e., $c_{uv} = 1$ for all $u, v \in V$ and 0 otherwise.

This case can be solved by a dynamic programming algorithm formulated in the same spirit of the previous section. The recursion uses two parameters m and k representing, respectively, the number of original nodes in V connected to the root of a subtree and the number of paths surviving in the same subtree.

Keeping the notation for subtrees introduced in Section 3.1, we define the following functions.

- $F_a(m,k)$ is the minimum total weight of the nodes from the set A to be removed from the subtree T_a in order to have node a connected to exactly m original nodes (including a itself) and k paths surviving in T_a.
- $G_{a_i}(m,k)$ is the minimum total weight of the nodes from the set A to be removed from the subtree $T_{a_{i,s}} = a + T_{a_i} + T_{a_{i+1}} + \cdots + T_{a_s}$ in order to have a connected to m original nodes of $T_{a_{i,s}}$ and k paths surviving in $T_{a_{i,s}}$.

We compute the values for F and G recursively for all $a \in V$, $m = 1, \ldots, n$, $k = 0, \ldots, n(n-1)/2$, as follows. Assume $F_a(m,k) = \infty$, $G_a(m,k) = \infty$ if $m < 0$ or $k < 0$.

$$F_a(m,k) = G_{a_1}(m,k) \qquad \text{for all non-leaf nodes } a \in V, \tag{4.6}$$

$$\begin{aligned} G_{a_i}(m,k) = \min\{ & w_c + \min\{F_{a_i}(p,q) + G_{a_{i+1}}(m, k-q) : p = 0, \ldots, \left|V(T_{a_i})\right|,\ q = 0, \ldots, k\}, \\ & \min\{F_{a_i}(p,q) + G_{a_{i+1}}[m-p, k-q-p(m-p)] : p = 0, \ldots, m,\ q = 0, \ldots, k\}\}, \end{aligned} \tag{4.7}$$

where c is the node connecting a to a_i. Equation (4.7) is written for all non-leaf nodes $a_i \in V$ with $i < s$ (non-rightmost subtrees).

For each rightmost subtree T_{a_s} we specify the initial condition

$$G_{a_s}(m,k) = \begin{cases} w_c + \min\{F_{a_s}(p,k) :\ p = 0, \ldots, \left|V(T_{a_s})\right|\} & \text{if } m = 1, \\ F_{a_s}(m-1, k-m+1) & \text{if } m > 1, \end{cases} \tag{4.8}$$

where c is the node connecting a to a_s, and for every leaf a:

$$F_a(m,k) = \begin{cases} 0 & \text{if } m = 1,\ k = 0, \\ \infty & \text{in all other cases.} \end{cases} \tag{4.9}$$

We use the same logic to explain equations (4.6)–(4.9) as we did in the previous section for equations (4.1)–(4.4).

For equation (4.6), note that $T_a = T_{a_{1,s}}$ for any non-leaf node $a \in V$.

For equation (4.7) note that the node c in A adjacent to a and a_i is the connecting node between the subtrees T_{a_i} and $T_{a_{i+1,s}}$. There are two cases to compute the value of

$G_{a_i}(m,k)$ based on the node c, either we remove the node c or we keep it. If the value of $G_{a_i}(m,k)$ is achieved from the expression $w_c + \min\{F_{a_i}(p,q) + G_{a_{i+1}}(m,k-q)\}$, then besides removing the optimal nodes from T_{a_i} and $T_{a_{i+1},s}$ we should also remove the node c and in this case a path surviving in $T_{a_i,s} = T_{a_i} + T_{a_{i+1},s}$ either completely belongs to T_{a_i} or to $T_{a_{i+1},s}$. According to the definition of $G_{a_i}(m,k)$, m original nodes are still connected to a and all nodes are inside the subtree $T_{a_{i+1},s}$, whereas at most $\left|V(T_{a_i})\right|$ original nodes can be connected to a_i. If q paths belong to T_{a_i} exactly $k-q$ paths belong to $T_{a_{i+1},s}$. Hence by definition of F and G the minimum total weight of the nodes from A removed from $T_{a_i,s}$ will be $G_{a_i}(m,k) = w_c + \min_{p,q}\{F_{a_i}(p,q) + G_{a_{i+1}}(m,k-q)\}$ where w_c corresponds the weight of the node c which connects a to a_i.

On the other hand if the node c is not removed, then a is connected to m original nodes of T_a and a path in $T_{a_i,s}$ can be either completely contained in one of T_{a_i}, $T_{a_{i+1},s}$, or partially contained in both subtrees because it passes through a. If a_i is connected to p original nodes of T_{a_i} and a is connected to $m-p$ original nodes in $T_{a_{i+1},s}$, the paths passing through a are exactly $p(m-p)$. If q paths survive in T_{a_i}, $k-q-p(m-p)$ paths survive in $T_{a_{i+1},s}$ and, by definition of F and G, $G_{a_i}(m,k) = \min_{p,q}\{F_{a_i}(p,q) + G_{a_{i+1}}[m-p, k-q-p(m-p)]\}$.

The initial condition (4.8) takes into account that, if the node c (which connects a to a_s) is removed ($m=1$), the k surviving paths of $T_{a_{s,s}} = a + T_{a_s}$ must belong entirely to T_{a_s}, hence the minimum possible weight for the nodes removed from $T_{a_{s,s}}$ will be $G_{a_s}(m,k) = w_c + \min_p\{F_{a_s}(p,k)\}$. On the other hand if the node c is not removed ($m>1$), in T_{a_s} we must have $m-1$ original nodes connected to a_s and $k-(m-1)$ surviving paths, since $m-1$ paths connect a to $m-1$ other nodes in $T_{a_{s,s}}$; thus $G_{a_s}(m,k) = F_{a_s}(m-1, k-m+1)$ follows.

The equation (4.9) says that since $a \in V$ is a leaf and it is not in the set A, the only possible condition is when $m=1, k=0$ and all other combinations of m and k are infeasible and are considered to have an infinite weight.

The optimal value, assuming the tree is rooted at node 1, is given by

$$\mathrm{OPT} = \min\{k\colon F_1(m,k) \le W, m = 1,\dots,n,\ k = 0,\dots,n(n-1)/2\}. \tag{4.10}$$

The optimal solution is recovered by backtracking.

Proposition 4.2.1. *CEDP on a tree with unit connection costs and arbitrary edge weights can be solved by recursion* (4.6)–(4.9) *in* $\mathcal{O}(n^7)$ *time.*

One could solve the CNEDP-1 and CNEDP-2 on a tree with subdivision approach.

Chapter 5

CNDP, CEDP, CNEDP-1, and CNEDP-2 with 0/1 connection costs

We will see in this chapter that CEDP, CNEDP-1, and CNEDP-2 are all NP-hard even under the assumptions that the node/edge weights are unitary and the connection costs are 0/1. We will also see that the polynomial-time algorithms that we presented in Chapter 3 actually apply to a more general case, in which the connection costs are 0/1 and have the following special structure: there exists $I \subseteq V$ such that

$$c_{uv} = \begin{cases} 1 & \text{if } u, v \in I \text{ and } u \neq v, \\ 0 & \text{otherwise.} \end{cases} \tag{5.1}$$

When the connection costs are of this form, we call them *square 0/1 connection costs*, because if these c_{uv}'s are represented via a 0/1 matrix, the support of the matrix is, up to permutation of rows and columns, a square (without one of its diagonals).

We mention that the case of square 0/1 connection costs has also some practical interest. Indeed, the role of the subset I is easy to understand: The elements in I are the nodes that we would really like to disconnect from each other, while the other nodes are part of the graph and it may be important to remove some of them to reduce the connectivity as much as possible, but they do not count in the evaluation of the objective function.

5.1 Hardness results for 0/1 connection costs

Di Summa, Grosso, and Locatelli [27] proved that CNDP on a tree is NP-hard even if the node weights are all equal to 1 and the connection costs are 0/1. Their proof is via a reduction from the decision version of (unweighted) Multicut in trees, which is known to be NP-complete [37] and is recalled here.

Problem 11 (MULTICUT IN TREES, decision version)**.** *Given a tree $G = (V, E)$, a list of pairs of nodes $(u_1, v_1), \ldots, (u_k, v_k)$, and a bound M, decide whether there exists $S \subseteq E$ with $|S| \leq M$ such that u_i and v_i are disconnected in $G - S$ for every $i \in \{1, \ldots, k\}$.*

It is immediate to see that MULTICUT IN TREES also reduces to each of CEDP, CNEDP-1, and CNEDP-2 on trees, as we observe below.

Observation 12. CEDP*,* CNEDP-1*, and* CNEDP-2 *are NP-hard even on a tree with unit node/edge weights and 0/1 connection costs.*

Proof. Given an instance of MULTICUT IN TREES with input as in Problem 11, we can reduce it to an instance of CEDP on the same tree, where (using the notation of Problem 3) $w_e = 1$ for all $e \in E$, $W = M$, $c_{u_i v_i} = 1$ for every $i \in \{1, \ldots, k\}$, and all other connection costs are equal to zero. Clearly a subset $S \subseteq E$ is feasible for the given instance of MULTICUT IN TREES if and only if the optimal value of the corresponding instance of CEDP is zero. This shows that CEDP is NP-hard even on a tree with unit node/edge weights and 0/1 connection costs. Since, as observed in the introduction, each of CNEDP-1 and CNEDP-2 subsumes CNDP and CEDP, we deduce the same result for CNEDP-1 and CNEDP-2. □

5.2 Solving CEDP and CNEDP-1 on a tree with 0/1 connection costs

In order to prove that CEDP and CNEDP-1 can be solved in polynomial time when the underlying graph is a tree with 0/1 connection costs, we begin by demonstrating a reduction from subdivision, which was previously discussed in Chapter 4, and which can be applied to any graph.

Lemma 5.2.1. CNEDP-1 *on a general graph G with square 0/1 connection costs can be polynomially reduced to* CNDP *with square 0/1 connection costs. Furthermore, when G is a tree the reduced instance is also defined on a tree.*

Proof. Let an instance of CNEDP-1 be given, with input as in Problem 4, where the connection costs are as in (5.1) for some $I \subseteq V$. Let $G' = (V', E')$ be the graph obtained by subdividing every edge of G.

We construct an instance of CNDP on G' with the following data. For $u, v \in V'$, the connection cost is

$$c'_{uv} = \begin{cases} 1 & \text{if } u, v \in I \text{ and } u \neq v, \\ 0 & \text{otherwise.} \end{cases}$$

Note that these are square 0/1 connection costs. The weights of the elements of $V' = V \cup E$ are defined by setting $w'_v = w_v$ for $v \in V$ and $w'_e = w_e$ for $e \in E$. The weight limit is the same as in the given instance of CNEDP-1, i.e., $W' = W$.

Given any $S \subseteq V \cup E = V'$, it is immediate to verify that $c(G - S) = c'(G' - S)$. Thus the optimal solutions of the instance of CNDP with square 0/1 connection costs constructed above are precisely the optimal solutions of the given instance of CNEDP-1. It is also clear that G' is a tree whenever G is a tree. □

Remark. It can be easily seen that the CNDP with general 0/1 connection costs on a complete graph in which we want to minimize the number of edges that survive after having removed at most K nodes is exactly equal to the CNDP with $L = 1$ on a general graph G in which we want to minimize the number of short paths that survive after having removed at most K nodes. Since the CNDP with $L = 1$ on a general graph G is NP-complete because of reduction from VERTEX COVER as we stated in Chapter 2, hence the CNDP with general 0/1 connection costs on a complete graph is also NP-complete. Furthermore, these two problems bridge the gap between the problem with no lower or upper constraint on the length of the connecting path and general nonnegative connection costs and the problem with unitary connection cost and $L = 1$.

5.3 Solving CNDP on a tree with 0/1 connection costs

We now show how to solve CNDP on a tree with square 0/1 connection costs in polynomial time, by means of a modification of the dynamic programming algorithm described in [27]. Because of the above lemma, this will also give a polynomial algorithm for CNEDP-1 on a tree with square 0/1 connection costs.

5.3.1 The case with 0/1 costs and unit weights

In this section we provide a polynomial algorithm for solving CNDP on trees where the connection costs c_{uv} are as in (5.1) for some important set of nodes $I \subseteq V$ and $w_v = 1$ for all $v \in V$. The nodes $v \in I$ are called important. In this scenario, the task requires reducing the number of connections that remain in the tree $T(V, E)$ after removing at most K nodes. Recall that in this problem we are interested in the number of connected pairs consisting of nodes in I.

To derive a dynamic programming algorithm, we introduce the following recursive functions:

- $F_a(m, k)$ = minimum number of connections surviving in T_a after k nodes have been removed from T_a and m important nodes are still connected to a (including a itself). Condition $m = 0$ indicates that no important node is connected with T_a.

- $G_{a_i}(m, k, t)$ = minimum number of connections surviving in the subtree $T_{a_{i,s}} = a + T_{a_i} + \cdots + T_{a_s}$ when k nodes are removed from $T_{a_{i,s}}$ and m important nodes are still

connected to a and $t \in \{0,1\}$, i.e., condition $t = 0$ indicates that a is removed from $T_{a_{i,s}}$ and $t = 1$ if a is not removed from $T_{a_{i,s}}$. As above, $m = 0$ indicates that no important node is connected with $T_{a_{i,s}}$.

We let the function values be infinity whenever the conditions cannot be satisfied.
We can state the following recursive relations:
For any non-leaf node $a \in V$ we have

$$F_a(m,k) = \min\{G_{a_1}(m,k,0),\ G_{a_1}(m,k,1)\}. \tag{5.2}$$

For any non-leaf node $a \in V$ and $i < s$ (non-rightmost subtrees) we also have

$$G_{a_i}(m,k,t) =$$
$$\begin{cases} \min\{F_{a_i}(p,q) + G_{a_{i+1}}(0,k-q,0) : p = 0,\dots,\left|V(T_{a_i})\right|, q = 0,\dots,k-1\} & \text{if } m = 0, t = 0, \\ \min\{F_{a_i}(0,q) + G_{a_{i+1}}(0,k-q,1) : q = 0,\dots,k\} & \text{if } m = 0,\ t = 1, \\ \min\{F_{a_i}(p,q) + G_{a_{i+1}}(m-p,k-q,1) + p(m-p) : & \\ \qquad p = 0,\dots,m,\ q = 0,\dots,k\} & \text{if } m > 0,\ t = 1, \\ \infty & \text{if } m > 0,\ t = 0. \end{cases} \tag{5.3}$$

For each rightmost subtree T_{a_s}, the initial conditions are the following:

$$G_{a_s}(m,k,t) =$$
$$\begin{cases} \infty & \text{if } (m = 0, k = 0, t = 0) \text{ or } (m > 0, t = 0), \\ \min\{F_{a_s}(p,k-1) : p = 0,\dots,|V(T_{a_s})|\} & \text{if } m = 0,\ k > 0, t = 0, \\ F_{a_s}(0,k) & \text{if } m = 0,\ t = 1, \\ F_{a_s}(m,k) + m & \text{if } m > 0, t = 1, a \notin I, \\ F_{a_s}(m-1,k) + (m-1) & \text{if } m > 0, t = 1, a \in I, \end{cases} \tag{5.4}$$

and, for every leaf a:

$$F_a(m,k) = \begin{cases} 0 & \text{if } (m = 0, k = 1, a \notin I) \text{ or } (m = 1, k = 0, a \in I), \\ \infty & \text{otherwise.} \end{cases} \tag{5.5}$$

Assuming that the tree T is rooted at node 1, the optimal value for the problem is given by

$$\text{OPT} = \min\{F_1(m,k) : m = 0,\dots,n,\ k = 0,\dots,K\}, \tag{5.6}$$

and an optimal solution can be reconstructed by backtracking.

Equation (5.2) is an immediate consequence of the observation that $T_a = T_{a_{1,s}}$ for every non-leaf node $a \in V$.

When a is a non-leaf node, in the formula for $G_{a_i}(m,k,t)$ (equation (5.3)) the case distinction is based on whether node a is removed ($t = 0$) or not ($t = 1$). In the former case, m is necessarily equal to zero. Expression $F_{a_i}(p,q)$ gives the minimum number of paths that survive in T_{a_i} when q nodes are removed from T_{a_i} and p important nodes of T_{a_i} are still connected to a_i. Since q nodes have been removed from T_{a_i}, exactly $k-q$ nodes (including a) must be removed from $T_{a_{i+1,s}}$. The minimum number of paths that survive in $T_{a_{i+1,s}}$ when $k-q$ nodes (including a) are removed from $T_{a_{i+1,s}}$ is given by $G_{a_{i+1}}(0,k-q,0)$. Thus the expression $F_{a_i}(p,q)+G_{a_{i+1}}(0,k-q,0)$ gives the minimum number of paths that survive in $T_{a_{i,s}}$ when q nodes are removed from T_{a_i} (and the other $k-q$ nodes are removed from $T_{a_{i+1,s}}$) and p important nodes of T_{a_i} are still connected to a_i. By taking the minimum over $p = 0,\ldots,|T_{a_i}|$ and $q = 0,\ldots,k-1$, we find the value of $G_{a_i}(0,k,0)$.

For the other case, i.e., when a is not removed ($t = 1$), one needs to know whether $m = 0$ or not. When $m = 0$, no important node is connected to the root a and also no important node is connected to the node a_i of T_{a_i}, because otherwise it would be connected to a. The expression $F_{a_i}(0,q) + G_{a_{i+1}}(0,k-q,1)$ gives the minimum number of paths that survive in $T_{a_{i,s}}$ when q nodes are removed from T_{a_i} and the other $k-q$ nodes are removed from $T_{a_{i+1,s}}$. By taking the minimum over $q = 0,\ldots,k$, we find the value of $G_{a_i}(0,k,1)$.

When $m > 0$, expression $F_{a_i}(p,q)$ gives the minimum number of paths that survive in T_{a_i} when q nodes are removed from T_{a_i} and p important nodes of T_{a_i} are still connected to a_i. Since q nodes have been removed from T_{a_i}, exactly $k-q$ nodes must be removed from $T_{a_{i+1,s}}$; and since p important nodes of T_{a_i} are still connected to a_i and thus to a, exactly $m-p$ important nodes of $T_{a_{i+1,s}}$ must remain connected to a. The minimum number of paths that survive in $T_{a_{i+1,s}}$ when $k-q$ nodes (without node a) are removed from $T_{a_{i+1,s}}$ and $m-p$ important nodes of $T_{a_{i+1,s}}$ are still connected to a is given by $G_{a_{i+1}}(m-p,k-q,1)$. Thus the expression $F_{a_i}(p,q) + G_{a_{i+1}}(m-p,k-q,1)$ gives the minimum number of paths that survive in T_{a_i} or $T_{a_{i+1,s}}$ when q nodes are removed from T_{a_i} and the other $k-q$ nodes are removed from $T_{a_{i+1,s}}$ and p important nodes of T_{a_i} are still connected to a_i, while $m-p$ important nodes of $T_{a_{i+1,s}}$ are still connected to a. Now we have to add the paths connecting important nodes of T_{a_i} to important nodes of $T_{a_{i+1,s}}$, i.e. $p(m-p)$ paths. This gives expression $F_{a_i}(p,q) + G_{a_{i+1}}(m-p,k-q,1) + p(m-p)$ of recursion (5.3). By taking the minimum over $p = 0,\ldots,m$ and $q = 0,\ldots,k$, we find the value of $G_{a_i}(m,k,1)$.

For a justification of (5.4), recall that $T_{a_{s,s}} = a + T_{a_s}$. There is a first distinction based on whether node a is removed ($t = 0$) or not ($t = 1$).The condition $m = 0$, $k > 0$, $t = 0$ means that we have to remove some nodes including the root a and hence the other $k-1$ nodes must be removed from the subtree T_{a_s} and so no important node is connected to the root a. For

the other case, i.e., when a is not removed ($t = 1$), one needs to know whether $m = 0$ or not. When $m = 0$, all of the k nodes are removed in T_{a_s}, and no important node in T_{a_s} can be connected to a_s, as otherwise it would be connected to a. When $m > 0$ and $a \notin I$, the value of $G_{a_s}(m, k, t)$ is obtained by requiring m important nodes in T_{a_s} connected to a_s. Finally, if $m > 0$ and $a \in I$, only $m - 1$ important nodes in T_{a_s} will be connected to a_s. Then we have to add all the connections of a to the nodes that are connected to a_s in the subtree.

Formula (5.5) reflects the fact that when the subtree consists of just a leaf, the decision to make is whether to remove ($k = 1$) the leaf or not ($k = 0$), and in both cases the number of path surviving in T_a is zero. The value of m can be one only if the leaf belongs to I and is not removed.

5.3.2 The case with $0/1$ costs and arbitrary node weights

This section considers the same problem as Section 5.3.1, but with general node weights instead of unit node weights.

Let $T(V, E)$ be a tree with $|V| = n$ nodes and let W be the available budget with weights w_v on the set of critical nodes. Given an important set of nodes $I \subseteq V$ with general $0/1$ connection costs c_{uv} for all $u, v \in I$, the problem is to remove a subset $S \subseteq V$ of total weight $w(S) \leq W$ such that the number of connected pairs of nodes in $T[V \setminus S]$ is as small as possible. The nodes $v \in I$ are called important.

We will calculate recursively the following values:

- $F_a(m, k)$ is the minimum total weight of the nodes to be removed from the subtree T_a in order to have node a connected to exactly m important nodes (including a itself) and k paths surviving in T_a.

- $G_{a_i}(m, k, t)$ is the minimum total weight of the nodes to be removed from the subtree $T_{a_{i,s}} = a + T_{a_i} + T_{a_{i+1}} + \cdots + T_{a_s}$ in order to have a connected to exactly m important nodes of $T_{a_{i,s}}$ and k paths surviving in $T_{a_{i,s}}$ and $t \in \{0, 1\}$, i.e., condition $t = 0$ indicates that a is removed from $T_{a_{i,s}}$ and $t = 1$ if a is not removed from $T_{a_{i,s}}$.

Furthermore, whenever the conditions in one of the above definitions cannot be satisfied, we set the value of the function to infinity. To simplify the recursive formulas below, it will be convenient to accept $k < 0$ in $F_a(m, k)$ and $G_{a_i}(m, k, t)$; in this case, we assume again that the function values are infinite.

At the end of the recursion, we can return the optimal value of the problem, assuming that the tree T is rooted at node 1, which is

$$\min\{k : F_1(m, k) \leq W,\ 0 \leq m \leq n,\ 0 \leq k \leq n(n-1)/2\}.$$

As usual in dynamic programming, an optimal solution can be reconstructed by backtracking.

We now provide the explicit formulas and then a justification for each of them. For every leaf a, we have

$$F_a(m,k) = \begin{cases} w_a & \text{if } m=0,\ k=0,\ a \in I, \\ 0 & \text{if } (m=0,\ k=0,\ a \notin I) \text{ or } (m=1,\ k=0,\ a \in I), \\ \infty & \text{otherwise,} \end{cases} \tag{5.7}$$

while for a non-leaf node a the formula is

$$F_a(m,k) = \min\{G_{a_1}(m,k,0),\ G_{a_1}(m,k,1)\}. \tag{5.8}$$

To calculate $G_{a_i}(m,k,t)$, where a is a non-leaf node, if $i=s$ we use the formula

$$G_{a_s}(m,k,t) = \begin{cases} \infty & \text{if } m>0, t=0, \\ w_a + \min\{F_{a_s}(p,k) : 0 \le p \le |V(T_{a_s}) \cap I|\} & \text{if } m=0, t=0, \\ F_{a_s}(0,k) & \text{if } m=0, t=1, \\ F_{a_s}(m,k) & \text{if } m>0, t=1, a \notin I, \\ F_{a_s}(m-1,k-m+1) & \text{if } m>0, t=1, a \in I, \end{cases} \tag{5.9}$$

while for $i<s$ the value of $G_{a_i}(m,k,t)$ is calculated recursively as follows:

$$G_{a_i}(m,k,t) = \begin{cases} \infty & \text{if } m>0,\ t=0, \\ \min\{F_{a_i}(p,q) + G_{a_{i+1}}(0,k-q,0) : p=0,\dots,|V(T_{a_i}) \cap I|, q=0,\dots,k\} & \text{if } m=0, t=0, \\ \min\{F_{a_i}(0,q) + G_{a_{i+1}}(0,k-q,1) : q=0,\dots,k\} & \text{if } m=0,\ t=1, \\ \min\{F_{a_i}(p,q) + G_{a_{i+1}}[m-p,k-q-p(m-p),1] : \\ \qquad p=0,\dots,m,\ q=0,\dots,k\} & \text{if } m>0,\ t=1. \end{cases} \tag{5.10}$$

We now give a justification for the above formulas. We denote by S a subset of nodes attaining the minimum value of $F_a(m,k)$ or $G_{a_i}(m,k,t)$ (depending on which formula we are illustrating).

Formula (5.7) reflects the fact that when the subtree consists of just a leaf, the decision to make is whether to remove the leaf or not, and in both cases k must be zero. The value of m can be one only if the leaf belongs to I and is not removed.

Equation (5.8) is an immediate consequence of the observation that $T_a = T_{a_{1,s}}$ for every non-leaf node $a \in V$.

When a is a non-leaf node, in the formula for $G_{a_s}(m,k,t)$ (equation (5.9)) there is a first distinction based on whether node a is remove ($t = 0$) or not ($t = 1$). In the former case, m is necessarily equal to zero, and we are on the second line of the formula, where the weight of node a is added to the minimum weight of a subset of nodes S' that should be removed from T_{a_s} to have k connected pairs consisting of nodes in I; note that p (the number of nodes in $V(T_{a_s} - S') \cap I$ that are connected to a_s) is arbitrary, as this value does not affect the number of nodes connected to a, as a is removed in this case. For the other case, i.e., when a is not removed ($t = 1$), one needs to know whether $m = 0$ or not. When $m = 0$, all of the k connected pairs are contained in T_{a_s}, and no node in $V(T_{a_s} - S) \cap I$ can be connected to a_s, as otherwise it would be connected to a, as well (third line of the formula). When, $m > 0$ and $a \notin I$, the value of $G_{a_s}(m,k,t)$ is obtained by requiring m nodes in $V(T_{a_s} - S) \cap I$ connected to a_s (as these nodes will be in turn connected to a) and a total connection cost of k in $T_{a_s} - S$, as $a \notin I$. Finally, if $m > 0$ and $a \in I$, only $m-1$ nodes in $V(T_{a_s} - S) \cap I$ will be connected to a_s, as a is also counted as connected to itself. Since these $m-1$ connections contribute to the total connection cost k in $T_{a_{i,s}} - S$, the total connection cost in $T_{a_s} - S$ must be $k - m + 1$.

Formula (5.10) is based on a similar case distinction, except that one does not need to know whether $a \in I$ or not. We only illustrate in detail the fourth case of the formula, i.e., $t = 1$ (i.e., node a is not removed) and $m > 0$. The formula is based on the observation that k must be the total cost of the connections surviving in each of the subtrees T_{a_i} and $T_{a_{i+1,s}}$, plus the cost of the connections between the two subtrees. This last term is given by $p(m-p)$, where p is the number of nodes in $V(T_{a_i} - S) \cap I$ connected to a_i and, consequently, $m - p$ is the number of nodes in $V(T_{a_{i+1,s}} - S) \cap I$ connected to a. Consequently, if q connected pairs survive in T_{a_i}, $k - q - p(m-p)$ survive in $T_{a_{i+1,s}}$. (For a correct interpretation of the formula, it is important to keep in mind that T_{a_i} is rooted at a_i, while $T_{a_{i+1,s}}$ is rooted at a.)

We obtain the following result.

Proposition 13. CNDP, CEDP, *and* CNEDP-1 *can be solved in polynomial time when the underlying graph is a tree and 0/1 square connection costs are given.*

Proof. By the above discussion, the proposed dynamic programming algorithm correctly solves CNDP when the underlying graph is a tree and 0/1 square connection costs are given. The polynomiality is immediate to check, as a, i, m, k and t can take only polynomially-many values, and the computation of each formula can be carried out in polynomial time.

Lemma 5.2.1 implies that CNEDP-1 can also be solved in polynomial time, under the same assumptions. Finally, the observation that CEDP is a special case of CNEDP-1 (see the introduction) proves the same result of CEDP. □

One can verify that the result for CNEDP-2 we have obtained in Proposition 3.5.2 extends to square 0/1 connection costs.

Chapter 6

Fixed number of leaves

In Chapter 2, we have seen that if the graph is a path, all the variants that we have considered have a closed-form solution. In this chapter, we investigate the case when we have general 0/1 connection costs on a path. We show that CNDP on a path can be solved in polynomial time, while Di Summa et al. [27] proved that CNDP on a tree is NP-hard even if the node weights are all equal to 1 and the connection costs are 0/1. Therefore the complexity differs between paths and trees in the case of nonnegative connection costs and unit node weights. One characterization of a path is that a path is a tree with two leaves. We then extend the number of leaves and show that CNDP, CEDP, and CNEDP-1 on a tree with arbitrary 0/1 connection costs and general nonnegative node/edge weights can be solved in polynomial time when the number of leaves of the tree is a constant.

6.1 The case with $0/1$ costs and unit node weights on paths

In this section we illustrate an algorithm for solving CNDP on paths when we are given general 0/1 connection costs c_{uv} for all u, v with unit weights w_v for all $v \in V$. For each $u, v \in V$, a connection is considered important if $c_{uv} = 1$. In this case the problem calls for minimizing the number of connections surviving in the path $P(V, E)$ after having removed at most K nodes.

Let $P(V, E)$ be a path with $|V| = n$ nodes and let the nodes are in order, i.e., $1, 2, \ldots, i, i+1, \ldots, n$. We are assuming that we have solved the sub-path problem optimally from node $i+1$ to node n. In order to solve the problem by dynamic programming, we define the following function:

- $F_i(k, t)$ = minimum number of connections surviving in the sub-path that begins at node i when k nodes are removed from the sub-path that starts at node i and ends at node n, with the minimum node that is removed at position t where $i \leq t \leq n+1$. Note that when no node is removed ($k = 0$) in the sub-path, then we define $t = n + 1$ to indicate

that there is no minimum node. Furthermore, if it is not possible to satisfy the above condition, then we define $F_i(k,t) = \infty$.

The values of F_i are calculated by traversing the path in postorder (from node n to node 1), by means of the following relations:

$$F_i(k,t) = \begin{cases} F_{i+1}(0,n+1) + \sum_{j=i+1}^{n} c_{ij} & \text{if } k = 0, \\ \min\{F_{i+1}(k-1,r) : r = i+1,\ldots,n+1\} & \text{if } k > 0,\ t = i, \\ F_{i+1}(k,t) + \sum_{j=i+1}^{t-1} c_{ij} & \text{if } k > 0,\ t > i. \end{cases} \tag{6.1}$$

The initial conditions for the last node are the following:

$$F_n(k,t) = \begin{cases} 0 & \text{if } (k=0,\ t=n+1) \text{ or } (k=1,\ t=n), \\ \infty & \text{otherwise.} \end{cases} \tag{6.2}$$

Recursion (6.1) can be interpreted as follows:

The case $k = 0$ means that we are not removing anything from the sub-path rooted at node i, i.e., we do not remove node i and we also don't remove any other node after node i. In this case we have to count all the important connections from node i to node n. Hence the number of paths that survive in the sub-path rooted at node i when we are not removing anything will be $F_i(0,0) = F_{i+1}(0,n+1) + \sum_{j=i+1}^{n} c_{ij}$.

If $k > 0$, then we have two options based on the node i has to be removed or not. The case $k > 0$, $t = i$ means that the node i has to be removed so that we are disconnecting all the connections that start at node i and therefore we have $k-1$ nodes left to remove in the sub-path. In this case we have to take $F_{i+1}(k-1,r)$ for every possible first node r varying from $i+1,\ldots,n+1$. Otherwise, $k > 0$, $t > i$ means that we are keeping the node i and t is the position of the minimum node that has to be removed and then we have to count all the important connections that go from node i to any node before node t.

Since $n \in V$ is the last node of the path, equation (6.2) says that either we remove the node n ($k = 1$) or we keep it ($k = 0$) and in both cases the number of important connections surviving is 0.

Assuming that the path P is rooted at node 1, the optimal value for the problem is given by

$$\text{OPT} = \min\{F_1(K,t) : t = 1,\ldots,n+1\}. \tag{6.3}$$

As usual in dynamic programming, an optimal solution can be recovered by backtracking.

Proposition 6.1.1. *CNDP on a path with square 0/1 connection costs and unit node weights can be solved by recursion* (6.1)–(6.2) *in* $\mathcal{O}(n^3K)$ *time.*

Proof. For each of the n nodes there are at most $\mathcal{O}(n)$ values for t and $\mathcal{O}(K)$ values for k; this gives $\mathcal{O}(n^2K)$ values of F to compute. The heaviest computation is that of equation (6.1) that requires at most $\mathcal{O}(n)$ steps. Hence in the worst case a number of operations bounded by $\mathcal{O}(n^3K)$ are required. □

6.2 The case with $0/1$ costs and arbitrary node weights on paths

This section addresses the same problem as the preceding one, but with general node weights rather than unit node weights.

Given the path $P = (V, E)$ with general $0/1$ connection costs and a budget W with weights w_v on the vertices, the problem is to remove a subset $S \subseteq V$ of total weight $w(S) \leq W$ such that the number of connected pairs of nodes in $P[V \setminus S]$ is as small as possible.

This case can be solved by a dynamic programming algorithm formulated in the same spirit of the previous section. The recursion uses two parameters k and t representing, respectively, the number of paths surviving in the sub-path and the position of the first node that is removed.

Keeping the notation introduced in the previous section, we define the following function.

- $F_i(k, t)$ is the minimum total weight of the nodes to be removed from the sub-path rooted at i, and t is the position of the minimum node that is removed where $i \leq t \leq n + 1$ and k important connections survive in the sub-path. Note that when no node is removed in the sub-path, then we define $t = n + 1$ to indicate that there is no minimum node. Furthermore, if it is not possible to satisfy the above condition, then we define $F_i(k, t) = \infty$.

We compute the values of F recursively for all $i \in V$, $k = 0, \ldots, n(n-1)/2$, $i \leq t \leq n+1$, as follows.

$$F_i(k,t) = \begin{cases} w_i + \min\{F_{i+1}(k,r) : r = i+1, \ldots, n+1\} & \text{if } t = i, \\ F_{i+1}(k - \sum_{j=i+1}^{t-1} c_{ij}, t) & \text{if } t > i. \end{cases} \tag{6.4}$$

The initial conditions for the last node are the following:

$$F_n(k,t) = \begin{cases} w_n & \text{if } k = 0,\ t = n, \\ 0 & \text{if } k = 0,\ t = n+1, \\ \infty & \text{in all other cases.} \end{cases} \tag{6.5}$$

Equation (6.4) can be interpreted as follows:

The case $t = i$ means that the minimum node which has to be removed is at position i. Since we are removing node i, we have to look at the sub-path starting at the node $i+1$ to get

k connections. Hence by definition of F the minimum total weight of the nodes to be removed from the sub-path rooted at i when the root is removed will be $F_i(k,t) = w_i + \min_r\{F_{i+1}(k,r)\}$ where w_i corresponds to the weight of the node i. On the other hand, if the root i of the sub-path is not removed, then all the nodes we have to remove is from the sub-path rooted at node $i+1$ and the position of the first node that has to be removed is at position t. Since we are keeping node i and the first node that is removed is at position t, the number of connections starting from node i to before node t is $\sum_{j=i+1}^{t-1} c_{ij}$ and the connections surviving in the sub-path rooted at node i will be $(k - \sum_{j=i+1}^{t-1} c_{ij})$. Thus by definition of F, $F_i(k,t) = F_{i+1}(k - \sum_{j=i+1}^{t-1} c_{ij}, t)$.

Equation (6.5) says that we get a connectivity of 0 by removing the last node n of the path as the first node. Otherwise, we get also a connectivity of 0 by removing nothing.

The optimal value, assuming the path is rooted at node 1, is given by

$$\mathrm{OPT} = \min\{k\colon F_1(k,t) \le W, k = 0,\dots,n(n-1)/2,\ t = 1,\dots,n+1\}. \tag{6.6}$$

The optimal solution is recovered by backtracking.

Proposition 6.2.1. *CNDP on a path with square 0/1 connection costs and arbitrary node weights can be solved by recursion* (6.4)–(6.5) *in* $\mathcal{O}(n^5)$ *time.*

Proof. For each node $i \in V$ there are at most $\mathcal{O}(n)$ values for t and $n(n-1)/2 + 1 = \mathcal{O}(n^2)$ values for k; this gives $\mathcal{O}(n^4)$ values of F to compute. The heaviest computation lies in equation (6.4), where $\mathcal{O}(n)$ values are possible for r. Hence in the worst case a number of operations bounded by $\mathcal{O}(n^5)$ are required.

□

The results derived for the CNDP on a path are summarised in the following table.

c_{uv}	w_v	complexity
$=0/1$	$=1$	solvable in $\mathcal{O}(n^3K)$
$=0/1$	≥ 0	solvable in $\mathcal{O}(n^5)$

Table 6.1: Complexity results for the CNDP on a path.

6.3 The case with $0/1$ costs and unit node weights on trees

In this section we show that if the graph is a tree with a fixed number of leaves and the connection costs take general $0/1$ values, each of CNDP, CEDP, and CNEDP-1 can be solved in polynomial time. Note that, unless $P = NP$, this result cannot hold for CNEDP-2,

as shown by Proposition 3.5.1. However, we will see that a polynomial-time algorithm can be found for this problem if, in addition, the node and edge weights are assumed to be unitary.

Let us first focus on CNDP on a tree with 0/1 connection costs and unit node weights. Let $T = (V, E)$ be a tree with n nodes and ℓ leaves. Given $u, v \in V$, we denote by $[u, v]$ the node set of the unique path joining u and v in T. Given $a \in V$ and $S \subseteq V$, we define the *boundary* of S with respect to a, and denote it by $B_a(S)$, as follows:

$$B_a(S) = \{v \in S : [a, v] \cap S = \{v\}\}.$$

Clearly, $|B_a(S)| \leq \ell$.

We will calculate the following values:

- $F_a(k, B)$ = minimum number of connections surviving in T_a when k nodes are removed from T_a and B is exactly the set of all first nodes on every path from the root to the leaves that is removed.

- $G_{a_i}(k, B)$ = minimum number of connections surviving in the subtree $T_{a_{i,s}} = a + T_{a_i} + \cdots + T_{a_s}$ when k nodes are removed from $T_{a_{i,s}}$ and B is exactly the set of all first nodes on every path from the root to the leaves that is removed.

The above function values are assumed to be infinite whenever the conditions cannot be satisfied. This happens, in particular, whenever B is not a set of the type $B_a(S)$ for any $S \subseteq V(T_a)$ (or $S \subseteq V(T_{a_{i,s}})$), for instance (but not only) when B is larger than the number of leaves of the subtree. (Indeed, we could use the number of leaves of T_a or $T_{a_{i,s}}$ as a bound for $|B|$, but this would slightly complicate the description of the algorithm).

If a is a leaf, we have

$$F_a(k, B) = \begin{cases} 0 & \text{if } (k = 0,\ B = \varnothing) \text{ or } (k = 1,\ B = \{a\}), \\ \infty & \text{otherwise,} \end{cases} \tag{6.7}$$

while if a is a non-leaf node the formula is

$$F_a(k, B) = G_{a_1}(k, B). \tag{6.8}$$

For a non-leaf node a we also have

$$G_{a_s}(k, B) = \begin{cases} \min\{F_{a_s}(k, B') : B' \subseteq V(T_{a_s}),\ |B'| \leq \ell\} & \text{if } B = \{a\}, \\ F_{a_s}(k, B) + \sum\{c_{av} : v \in V(T_{a_s}),\ [a, v] \cap B = \varnothing\} & \text{if } a \notin B, \\ \infty & \text{otherwise,} \end{cases} \tag{6.9}$$

and, for $i < s$,

$$G_{a_i}(k, B) =$$
$$\begin{cases} \min\{F_{a_i}(q, B') + G_{a_{i+1}}(k-q, \{a\}) : 0 \le q \le k-1,\ B' \subseteq V(T_{a_i}),\ |B'| \le \ell\} & \text{if } B = \{a\}, \\ \min\{F_{a_i}(q, B \cap V(T_{a_i})) + G_{a_{i+1}}(k-q, B \cap V(T_{a_{i+1,s}}) + \sum\{c_{uv} : \\ \qquad u \in V(T_{a_i}),\ v \in V(T_{a_{i+1,s}}),\ [u,v] \cap B = \varnothing\} : 0 \le q \le k\} & \text{if } a \notin B, \\ \infty & \text{otherwise,} \end{cases} \tag{6.10}$$

If we denote by 1 the root node of the tree, the optimal value is given by

$$\min\{F_1(K, B) : B \subseteq V,\ |B| \le \ell\}.$$

The justification of (6.7) and (6.8) is immediate to check.

In formula (6.9), the first and third cases express the fact that if $a \in B$ and B is a set of the form $B_a(S)$ for some $S \subseteq V(T_a)$, then necessarily $B = S = \{a\}$. In this case, a is removed from T_a and we are left with the subproblem on T_{a_s}. (In this subproblem the boundary set B' is arbitrary, as it does not affect the fact that $B = \{a\}$.) On the other hand, if $a \notin B$ and $B = B_a(S)$ for some $S \subseteq V(T_a)$, then $S \subseteq V(T_{a_s})$ and $B = B_{a_s}(S)$. We then obtain the second case of the formula, where the term $\sum\{c_{av} : v \in V(T_{a_s}),\ [a,v] \cap B = \varnothing\}$ counts the number of nodes connected to a in $T_{a_s} - S$.

Formula (6.10) is based on a similar argument. We only remark some points. When $B = S = \{a\}$, two subproblems on T_{a_i} and $T_{a_{i+1,s}}$ are created. Since a is removed from $T_{a_{i,s}}$, the boundary set of the first subproblem is an arbitrary B' (as this does not affect the fact that $B = \{a\}$), while in the second subproblem the boundary set must be $\{a\}$ (as a is the root of $T_{a_{i+1,s}}$). When $a \notin B$, we use the fact that if $B = B_a(S)$ for some $S \subseteq V(T_{a_{i,s}})$, then $B_a(S \cap V(T_{a_i})) = B \cap V(T_{a_i})$ and $B_a(S \cap V(T_{a_{i+1,s}}) = B \cap V(T_{a_{i+1,s}})$. The term $\sum\{c_{uv} : u \in V(T_{a_i}),\ v \in V(T_{a_{i+1,s}}),\ [u,v] \cap B = \varnothing\}$ count the number of surviving connections between the two subtrees.

6.4 The case with $0/1$ costs and arbitrary node weights on trees

In this section we manage to derive a polynomial time algorithm when the number of leaves ℓ is a constant also for the case with arbitrary weights w_v assigned to the nodes $v \in V$. The problem is to remove a subset $S \subseteq V$ of total weight $w(S) \le W$ such that the number of connected pairs of nodes in $T[V \setminus S]$ is as small as possible.

Keeping the notation for subtrees introduced in the previous section, we introduce the following functions.

- $F_a(k, B)$ is the minimum total weight of the nodes to be removed from the subtree T_a and B is exactly the set of all first nodes on every path from the root to the leaves that is removed and k connections surviving in T_a.
- $G_{a_i}(k, B)$ is the minimum total weight of the nodes to be removed from the subtree $T_{a_{i,s}} = a + T_{a_i} + T_{a_{i+1}} + \cdots + T_{a_s}$ and B is exactly the set of all first nodes on every path from the root to the leaves that is removed and k connections surviving in $T_{a_{i,s}}$.

We set the value $F_a(k, B) = \infty$ or $G_{a_i}(k, B) = \infty$ when no feasible solution exists. Also, it is convenient to set the above values to infinity when $k < 0$. Using similar recursive arguments of the previous dynamic program, we state the following recursions:

If a is a leaf, we have

$$F_a(k, B) = \begin{cases} w_a & \text{if } k = 0 \text{ and } B = \{a\}, \\ 0 & \text{if } k = 0 \text{ and } B = \varnothing, \\ \infty & \text{otherwise,} \end{cases} \tag{6.11}$$

while if a is a non-leaf node the formula is

$$F_a(k, B) = G_{a_1}(k, B). \tag{6.12}$$

For a non-leaf node a we also have

$$G_{a_s}(k, B) = \begin{cases} w_a + \min\{F_{a_s}(k, B') : B' \subseteq V(T_{a_s}),\, |B'| \le \ell\} & \text{if } B = \{a\}, \\ F_{a_s}\big(k - \sum\{c_{av} : v \in V(G_{a_s}),\, [a, v] \cap B = \varnothing\}, B\big) & \text{if } a \notin B, \\ \infty & \text{otherwise,} \end{cases} \tag{6.13}$$

and, for $i < s$,

$$G_{a_i}(k, B) = \begin{cases} \min\{F_{a_i}(q, B') + G_{a_{i+1}}(k - q, \{a\}) : 0 \le q \le k,\, B' \subseteq V(T_{a_i}),\, |B'| \le \ell\} & \text{if } B = \{a\}, \\ \min\{F_{a_i}(q, B \cap V(T_{a_i})) + G_{a_{i+1}}(k - q - \sum\{c_{uv} : u \in V(T_{a_i}), \right. & \\ \qquad \left. v \in V(T_{a_{i+1,s}}),\, [u, v] \cap B = \varnothing\}, B \cap V(T_{a_{i+1,s}})) : 0 \le q \le k\} & \text{if } a \notin B, \\ \infty & \text{otherwise.} \end{cases} \tag{6.14}$$

If we denote by 1 the root node of the tree, the optimal value is given by

$$\min\{k : F_1(k, B) \le W,\, 0 \le k \le n(n-1)/2,\, B \subseteq V,\, |B| \le \ell\}.$$

We obtain the following result.

Proposition 14. CNDP, CEDP, *and* CNEDP-1 *can be solved in polynomial time when the underlying graph is a tree with a constant number of leaves and the connection costs are 0/1.*

Proof. For CNDP the polynomiality follows from the above algorithm, after observing that the number of boundary sets B to consider is $O(n^\ell)$, which is polynomial when ℓ is constant. To derive a polynomial-time dynamic programming algorithm for CNEDP-1, one can use the subdivision approach described in the proof of Lemma 5.2.1. (We omit the details.) The polynomiality of CEDP then follows because, as already observed, this problem is a special case of CNEDP-1. □

For CNEDP-2, we need to restrict to unit weights.

Proposition 15. CNEDP-2 *can be solved in polynomial time when the underlying graph is a tree with a constant number of leaves, the node and edge weights are unitary, and the connection costs are 0/1.*

Proof. A polynomial-time dynamic programming algorithm can be obtained by combining the approach described in this section with that developed in Section 3.5. The basic idea is to define $F_a(k_V, k_E, B)$ as the minimum cardinality of a subset $S \subseteq V(T_a) \cup E(T_a)$ such that $|S \cap V| \leq k_V$, $|S \cap E| \leq k_E$, and $B_a(S) = B$, and similarly for $G_{a_i}(k_V, k_E, B)$. We omit the details. □

The results derived for the CNDP on a tree with a fixed number of leaves are summarised in the following table.

c_{uv}	w_v	complexity
$=0/1$	$=1$	solvable in $\mathcal{O}(n^3 K n^\ell)$
$=0/1$	≥ 0	solvable in $\mathcal{O}(n^5 n^\ell)$

Table 6.2: Complexity results for the CNDP on a tree with a fixed number of leaves.

Chapter 7

IP formulations of CNDP, CEDP, CNEDP-1, and CNEDP-2

In this chapter we focus on exact integer programming solution methods to solve critical node and/or edge detection problems on general graphs. The mathematical formulation introduced in [8] is designed to tackle the CNDP for the case when the total number of pairwise connections is used as the connectivity measure and when the deletion costs are set equal to one. The authors in [8] present an integer programming (IP) formulation of CNDP using triangle inequalities that enforce transitive connectivity relationships between nodes in the graph, i.e., if node i is connected to node j and node j is connected to node k, then node i should also be connected to node k. Note that this formulation can be easily adapted to use the weighted pairwise connectivity or to include deletion costs different than one. In this chapter, we provide a modified mathematical formulation based on the linear integer programming model for the CNDP described in [8], which we call restricted CNDP. We also derive IP models from the proposed restricted CNDP which provides optimal solutions for the CEDP, CNEDP-1, and CNEDP-2.

7.1 IP formulations for the restricted CNDP

Let $G = (V, E)$ be an undirected graph with a set of $|V| = n$ nodes and a set of edges $E \subseteq V \times V$. Define a restricted critical node detection problem (RCNDP) in which we are also given the set of allowed vertices $A \subseteq V$ and the problem is to disconnect the graph as much as possible by removing K nodes in the allowed set A. The RCNDP becomes the standard CNDP when $A = V$. This restricted problem has been treated also in Chapter 4.

The optimal solution of the RCNDP can be determined by using an integer programming (IP) formulation. The IP formulation of the RCNDP is described below.

Let the input data of the problem be a nonnegative connection cost c_{ij} for each pair of distinct nodes $i, j \in V$, a weight $w_i \geq 0$ for each $i \in V$ and a bound W.

For any node $i \in V$, the binary variable v_i is defined as

$$v_i = \begin{cases} 1, & \text{if node } i \text{ is deleted from the graph} \\ 0, & \text{otherwise.} \end{cases} \tag{7.1}$$

Introduce a set of connectivity variables between pairs of nodes $i, j \in V, i \neq j$:

$$u_{ij} = \begin{cases} 1, & \text{if } i, j \in V \setminus A, i \neq j \text{ and } i, j \text{ are in the same component of } G(V \setminus A) \\ 0, & \text{otherwise.} \end{cases} \tag{7.2}$$

Before we continue, note that the set of u_{ij} variables should necessarily satisfy the following condition:

$$u_{ij} = u_{ji},\ i, j \in V, i \neq j.$$

The restricted critical node detection problem can be formulated as:

$$\text{(RCNDP)} \qquad \text{Minimize} \sum_{i,j \in V,\ i<j} c_{ij} u_{ij} \tag{7.3}$$

subject to

$$u_{ij} + v_i + v_j \geq 1,\ \forall ij \in E, \tag{7.4}$$

$$u_{ij} + u_{jk} - u_{ki} \leq 1,\ \forall\ i \in V,\ \forall\ jk \in E,\ j < k, \tag{7.5}$$

$$u_{ik} + u_{kj} - u_{ji} \leq 1,\ \forall\ i \in V,\ \forall\ jk \in E,\ j < k, \tag{7.6}$$

$$\sum_{i \in A} w_i v_i \leq W, \tag{7.7}$$

$$v_i = 0,\ \forall i \notin A, \tag{7.8}$$

$$u_{ij} \in \{0, 1\},\ \forall i, j \in V, \tag{7.9}$$

$$v_i \in \{0, 1\},\ \forall i \in V, \tag{7.10}$$

where the objective function (7.3) minimizes the weighted sum of pairwise connections. Constraint (7.4) means that if nodes i and j are in different components and there is an edge between them, then one of them should be deleted. As shown in [60], constraints (7.5) and (7.6) altogether indicate that if i is connected to j and j is connected to k, then i must also be connected with k. Constraint (7.7) sets the upper bound of the budget. Constraint (7.8) says that only the allowed vertices can be removed. Finally, (7.9) and (7.10) define the proper domains for the variables used.

7.2 IP formulations for the CEDP

Let $G = (V, E)$ be an undirected graph with $|V| = n$ nodes. Now we apply the subdivision process on G which has been described in Chapter 4 and we have created the graph $G' = (V', E')$ in which all the edges are subdivided. Now we consider the restricted CNDP on G' with the allowed set $A = V' \setminus V$, i.e., the set of newly created vertices and the connection cost c_{ij} takes the value of 1 if $i, j \in V, i \neq j$ and 0 otherwise, i.e., we only count the connections between the original vertices in G. Then the problem of removing at most K edges from G becomes the problem of removing at most K nodes from G' but here we are allowed to remove K nodes only from the set A of newly created nodes and the IP formulations for the CEDP is the same as described in equations (7.3)–(7.10).

7.3 IP formulations for the CNEDP-2

Let $G = (V, E)$ be an undirected graph and let W_V and W_E be the available budgets for removing the cost of nodes and edges respectively from the graph G. Construct a graph $G' = (V', E')$ with the subdivision as mentioned in Chapter 4 and we consider the restricted CNDP on G' with the allowed set $A = V' \setminus V$, i.e., the set of newly created vertices. We have now two kinds of nodes and we call them the set of original vertices which are in the set V and the set of newly created vertices which are in the set A. In the CNEDP-2, we are allowed to remove nodes from both of them but they are counting in different budgets. Once the subdivision has been made, the vertices in V keep the same cost as the original vertices and the new vertices in A inherit the cost of the edges from which they are subdividing because every vertex subdivides an edge and therefore the edges in G' has no cost anymore.

The mixed problem for two separate budgets can be defined as:

$$\text{(CNEDP-2)} \qquad \text{Minimize} \sum_{i,j \in V,\ i<j} c_{ij} u_{ij} \tag{7.11}$$

subject to

$$u_{ij} + v_i + v_j \geq 1,\ \forall ij \in E', \tag{7.12}$$

$$u_{ij} + u_{jk} - u_{ki} \leq 1,\ \forall\ i \in V',\ \forall\ jk \in E',\ j < k, \tag{7.13}$$

$$u_{ik} + u_{kj} - u_{ji} \leq 1,\ \forall\ i \in V',\ \forall\ jk \in E',\ j < k, \tag{7.14}$$

$$\sum_{i \in V} w_i v_i \leq W_V, \tag{7.15}$$

$$\sum_{i \in A} w_i v_i \leq W_E, \tag{7.16}$$

$$u_{ij} \in \{0,1\},\ \forall i,j \in V', \tag{7.17}$$

$$v_i \in \{0,1\},\ \forall i \in V'. \tag{7.18}$$

Constraint (7.15) says that sum of the weights of the original nodes cannot exceed the budget W_V and Constraint (7.16) says that sum of the weights of the newly created nodes is at most W_E.

7.4 IP formulations for the CNEDP-1

The IP formulations for the CNEDP-1 is almost same as the case of the CNEDP-2. Instead of equations (7.15) and (7.16), there will be a single constraint $\sum_{i \in V'} w_i v_i \leq W$ which says that sum of the weights of the original nodes and the newly created nodes that we remove altogether cannot exceed the budget W.

Chapter 8

Distance-based Critical Node/Edge Detection Problem

In this chapter, we study a special case of the Critical Node/Edge Detection Problem, the so-called Distance-based Critical Node Detection Problem (D-CNDP) and the Distance-based Critical Edge Detection Problem (D-CEDP) as introduced in [82], where the distances between node pairs impact on the objective function. In this context we analysis the following different classes of D-CNDP on trees in which the objectives are to:

1. Minimize the number of node pairs connected by a path of length $\leq L$;
2. Maximize the number of node pairs connected by a path of length $\geq L$;
3. Maximize the number of node pairs connected by a path of length $\leq L$;
4. Minimize the number of node pairs connected by a path of length $\geq L$.

In this chapter, for the sake of simplicity, we propose dynamic programming algorithms for Class 1 and Class 2 mentioned above. We would like to mention that one can obtain dynamic programming algorithm for Class 3 from Class 1 just putting maximization instead of minimization and similarly, for Class 4 from Class 2 by changing minimization instead of maximization. We note that we have developed the aforesaid dynamic programming algorithms at the beginning of my PhD, and then after one year we found that the authors in [7] proposed dynamic programming algorithms for some special cases of D-CNDP including the particular variant of Class 1.

We also present a structural property of Class 2 D-CNDP on trees which says that we recursively remove only leaves in an optimal solution. After that we move to D-CEDP over trees and present dynamic programming algorithm and integer programming formulation for the variant of Class 1.

8.1 D-CNDP on trees for Class 1

In this context, the problem is to minimize the amount of node-pairs remaining connected by a path of length at most L, for some natural integer L, that survive in a tree $T(V, E)$ after at most K nodes have been removed.

Recall the notation for subtrees. Given the tree $T(V, E)$ with $|V| = n$, let T_a be the subtree of T rooted at $a \in V$. If a is not a leaf of T, let $T_{a_1}, \ldots, T_{a_s}$ be the subtrees of T_a rooted at the children nodes $a_1, \ldots, a_s$ respectively, where s depends on a. Let also $|T_a|$ be the number of nodes in T_a. The functions below are defined in order to solve the problem using dynamic program.

- $F_a(m_0, m_1, \cdots, m_{L-1}, k)$ = minimum number of paths of length $\leq L$ surviving in T_a when k nodes are removed from T_a and $m_0, m_1, \cdots, m_{L-1}$ represents the number of nodes surviving in T_a having distance $0, 1, 2, \cdots, (L-1)$ respectively from a. Note that condition $m_0 = 0$ indicates that a is removed from T_a and $m_0 = 1$ if a is not removed from T_a.

- $G_{a_i}(m_0, m_1, \cdots, m_{L-1}, k)$ = minimum number of paths of length $\leq L$ surviving in the subtree $T_{a_{i,s}} = a + T_{a_i} + \cdots + T_{a_s}$ when k nodes are removed from $T_{a_{i,s}}$ and $m_0, m_1, \cdots, m_{L-1}$ represents the number of nodes surviving in $T_{a_{i,s}}$ having original distance $0, 1, 2, \cdots, (L-1)$ respectively from a. As above, $m_0 = 0$ indicates that a is removed from $T_{a_{i,s}}$.

We let the function values be infinity whenever no feasible solution exists. The values for F and G can be computed by traversing the tree in postorder (from leaves to root), by means of the following relations:

$$F_a(m_0, m_1, \cdots, m_{L-1}, k) = G_{a_1}(m_0, m_1, \cdots, m_{L-1}, k) \quad \text{for any non-leaf node } a \in V; \quad (8.1)$$

$$G_{a_i}(m_0, m_1, \cdots, m_{L-1}, k) =$$
$$\begin{cases} \min\{F_{a_i}(m'_0, m'_1, \cdots, m'_{L-1}, q) + G_{a_{i+1}}(0, m''_1, \cdots, m''_{L-1}, k-q) : \\ \qquad m'_0 \in \{0,1\},\ m'_j = 0, \ldots, \left|V(T_{a_{i,j}})\right| - 1,\ m''_j = m_j - m'_{j-1}, \\ \qquad q = 0, \ldots, k-1\} & \text{if } m_0 = 0, \\ \min\{F_{a_i}(m'_0, m'_1, \cdots, m'_{L-1}, q) + G_{a_{i+1}}(1, m''_1, \cdots, m''_{L-1}, k-q) \\ \qquad + \sum_{l=1}^{L} \sum_{j=0}^{l-1} m'_j m''_{l-1-j} : m'_0 \in \{0,1\}, \\ \qquad m'_j = 0, \ldots, \left|V(T_{a_{i,j}})\right| - 1,\ m''_j = m_j - m'_{j-1},\ q = 0, \ldots, k\} & \text{if } m_0 = 1, \end{cases} \quad (8.2)$$

for any non-leaf node $a \in V$ where $i < s$ and $\left|V(T_{a_{i,j}})\right|$ denotes the number of nodes in T_{a_i} at distance j from a_i for every $j = 1, \ldots, L-1$.

For each rightmost subtree T_{a_s} we specify the following initial conditions:

$$G_{a_s}(m_0, m_1, \cdots, m_{L-1}, k) =$$
$$\begin{cases} \infty & \text{if } m_0 = k = 0, \\ \min\{F_{a_s}(m'_0, m'_1, \cdots, m'_{L-1}, k-1) : m'_0 \in \{0,1\}, & \\ \qquad m'_j = 0, \ldots, \left|V(T_{a_{s,j}})\right| - 1\} & \text{if } m_0 = 0,\ k > 1, \\ \min\{F_{a_s}(m'_0, m'_1, \cdots, m'_{L-1}, k) + m'_0 + m'_1 + \cdots + m'_{L-1} : & \\ \qquad m'_0 \in \{0,1\}, m'_j = 0, \ldots, \left|V(T_{a_{s,j}})\right| - 1\} & \text{if } m_0 = 1, \end{cases} \quad (8.3)$$

while, for every leaf $a \in V$ we have that:

$$F_a(m_0, m_1, \cdots, m_{L-1}, k) = \begin{cases} 0 & \text{if } (m_0 = \cdots = m_{L-1} = 0, k = 1) \text{ or} \\ & (m_0 = 1, m_1 = \cdots = m_{L-1} = 0, k = 0), \\ \infty & \text{otherwise.} \end{cases} \quad (8.4)$$

We now give a justification for the above formulas. Formula (8.1) is immediate because it follows from the fact that $T_a = T_{a_{1,s}}$ for every non-leaf node $a \in V$.

In the formula for $G_{a_i}(.)$ (equation (8.2)), the separation is based on whether node a is removed ($m_0 = 0$) or not ($m_0 = 1$) from the tree. The first case ($m_0 = 0$) corresponds to having $a \in S$. In this situation, we take the sum of the optimal values that we can obtain in each of the two subtrees T_{a_i} and $T_{a_{i+1,s}}$. For the second case ($m_0 = 1$), in which $a \notin S$, we take again the sum of the optimal values in each of the two subtrees T_{a_i} and $T_{a_{i+1,s}}$; and since node a is surviving, a path in $T_{a_{i,s}}$ is passing through a and in this case we have to add the connections between the two subtrees which are connected by a path of length at most L (which is given by the third term).

Formula (8.3) is based on a similar argument as $T_{a_{s,s}} = a + T_{a_s}$.

Equation (8.4) reflects the fact that when the subtree consists of just a leaf, the decision to make is whether to remove the leaf ($k = 1$) or not ($k = 0$), and in both cases the number of paths survive in T_a is 0.

Assuming that the tree T is rooted at node 1, the optimal value for the problem is given by

$$\text{OPT} = \min\{F_1(m_0, \cdots, m_{L-1}, K) : m_0 \in \{0,1\},\ m_1, \cdots, m_{L-1} \geq 0,\ \sum_{j=1}^{L-1} m_j \leq n\}, \quad (8.5)$$

and the optimal solution is recovered by backtracking. We get the following proposition.

Proposition 8.1.1. *D-CNDP on trees for Class 1 with unit costs and unit node weights can be solved by recursion* (8.1)–(8.4) *in* $\mathcal{O}(n^{3L-2}K^2)$ *time.*

Proof. For each node $a \in V$ there are at most $\mathcal{O}(n^L)$ values for $m_0, m_1, \cdots, m_{L-1}$ and $\mathcal{O}(K)$ values for k; this gives $\mathcal{O}(n^{2L-1}K)$ values of F and G to compute. The heaviest computation is that of equation (8.2) that requires at most $\mathcal{O}(n^{L-1}K)$ steps. Hence in the worst case a number of operations bounded by $\mathcal{O}(n^{3L-2}K^2)$ are required. □

8.2 D-CNDP on trees for Class 2

The goal in this circumstance is to maximize the number of node-pairs still connected by a path of length at least L, for some natural number L, that survive in a tree $T(V, E)$ after removing at most K nodes.

For convenience we denote by $D(T)$ the diameter of the tree T, by $D(T_{a_{i,s}})$ the length of the longest path from a node in T_{a_i} to a node in $T_{a_{i+1,s}}$, by $D(T_{a_s})$ the length of the longest path from node a to a node in T_{a_s}. Let $L' = D(T) - L$ where L' is fixed. In order to solve the problem by dynamic programming, we define the following functions.

- $F_a(m_0, m_1, \cdots, m_{L'-1}, k)$ = maximum number of paths of length $\geq L'$ surviving in T_a when k nodes are removed from T_a and for every j, m_j is the number of nodes whose distance from the root a is $D(T_a) - j$ (for instance, m_0 is the number of nodes farthest from the root, m_1 is the number of nodes on the level before and so on).

- $G_{a_i}(m_0, m_1, \cdots, m_{L'-1}, k)$ = maximum number of paths of length $\geq L'$ surviving in the subtree $T_{a_{i,s}} = a + T_{a_i} + \cdots + T_{a_s}$ when k nodes are removed from $T_{a_{i,s}}$ and for every j, m_j is the number of nodes whose distance from the root a is $D(T_{a_{i,s}}) - j$.

We let the function values be infinity whenever the conditions cannot be satisfied. The values for F and G can be computed by traversing the tree in postorder (from leaves to root), by means of the following relations:

$$F_a(m_0, m_1, \cdots, m_{L'-1}, k) = G_{a_1}(m_0, m_1, \cdots, m_{L'-1}, k) \quad \text{for any non-leaf node } a \in V; \tag{8.6}$$

$$G_{a_i}(m_0,\cdots,m_{L'-1},k)=$$
$$\begin{cases} 0 & \text{if } D(T_{a_{i,s}})<D-L', \\ \max\{F_{a_i}(m'_0,\cdots,m'_{L'-1},q)+G_{a_{i+1}}(m''_0,\cdots,m''_{L'-1},k-q) & \\ \quad +\sum_{l=0}^{t}\sum_{j=0}^{t+l} m'_j m''_{t+l-j} : q=0,\ldots,k; m'_0,\cdots,m'_{L'-1}\geq 0; & \\ \quad m''_0,\cdots,m''_{L'-1}\geq 0; m'_0+\cdots+m'_{L'-1}\leq |T_{a_i}|; & \\ \quad m''_0+\cdots+m''_{L'-1}\leq \left|T_{a_{i+1,s}}\right|\} & \text{if } D(T_{a_{i,s}})=D-L'+t \text{ for some } t=0,\cdots,L', \end{cases} \tag{8.7}$$

for any non-leaf node $a\in V$ and $i<s$.

The initial conditions on each leaf a and on each rightmost subtree T_{a_s} are the following:

$$F_a(m_0,\cdots,m_{L'-1},k)=\begin{cases} 0 & \text{if } (m_0=\cdots=m_{L'-1}=0,k=1) \text{ or} \\ & (m_0=1,m_1=\cdots=m_{L'-1}=0,k=0), \\ -\infty & \text{otherwise,} \end{cases} \tag{8.8}$$

$$G_{a_s}(m_0,\cdots,m_{L'-1},k)=$$
$$\begin{cases} -\infty & \text{if } m_0=\cdots=m_{L'-1}=k=0, \\ 0 & \text{if } D(T_{a_s})<D-L', \\ \max\{F_{a_s}(m'_0,\cdots,m'_{L'-1},k-1), F_{a_s}(m'_0,\cdots,m'_{L'-1},k) & \\ \quad +\sum_{j=0}^{t-1} m'_j : m'_0,\cdots,m'_{L'-1}\geq 0; & \\ \quad m'_0+\cdots+m'_{L'-1}\leq |T_{a_s}|\} & \text{if } D(T_{a_s})=D-L'+t \text{ for some } t=0,\cdots,L'. \end{cases} \tag{8.9}$$

Formula (8.8) is immediate, while (8.6) follows from the fact that $T_a=T_{a_{1,s}}$ for every non-leaf node $a\in V$.

In equation (8.7), we have to find paths of length $D-L', D-L'+1,\cdots,D$ that are surviving in T_{a_i} or in $T_{a_{i+1},s}$ or in $T_{a_{i,s}}=T_{a_i}+T_{a_{i+1,s}}$. In order to compute the overall number of nodes connected to a at a minimum distance of $D-L'$ in $G_{a_i}(.)$, we have to consider the maximum number of paths that survive in T_{a_i} when q nodes are removed from T_{a_i} plus the maximum number of paths that survive in $T_{a_{i+1,s}}$ when $k-q$ nodes are removed from $T_{a_{i+1,s}}$. Also, the third term should be added if $T_{a_{i,s}}$ contains a path of length at least $D-L'$ that intersects both T_{a_i} and $T_{a_{i+1},s}$.

The justification of formula (8.9) is similar to (8.7) and in this case one also need to know the node a_s is removed or not. Considering the removal of the node a_s, we have to take the better of two possibilities, which correspond to the two arguments of the outer maximum.

If we denote by 1 the root node of the tree T, the optimal value for the problem is given by

$$\text{OPT} = \max\{F_1(m_0, \cdots, m_{L'-1}, K) : m_0, m_1, \cdots, m_{L'-1} \geq 0, \sum_{j=0}^{L'-1} m_j \leq n\}, \tag{8.10}$$

and the optimal solution is recovered by backtracking. We obtain the following result.

Proposition 8.2.1. *D-CNDP on trees for Class 2 with unit costs and unit node weights can be solved by recursion* (8.6)–(8.9) *in* $\mathcal{O}(n^{3L'+1}K^2)$ *time.*

Proof. The proposed dynamic program, the number of functions $F_a(.)$ and $G_a(.)$ can be bounded by $\mathcal{O}(n^{2L'+1}K)$ to compute for each node $a \in V$. The heaviest computation is that of equation (8.7) that requires at most $\mathcal{O}(n^{L'}K)$ steps. The overall complexity is thus $\mathcal{O}(n^{3L'+1}K^2)$. □

We also established the following result:

Theorem 8.2.2. *For the problem of maximizing the number of paths of length at least L, there is an optimal solution which can be obtained by recursively removing a leaf from the residual graph.*

Proof. The problem calls for maximizing the number of paths of length $\geq L$ surviving in a tree $T = (V, E)$ after having removed at most K nodes. Let $v_1, v_2, \cdots, v_k$ be nodes removed in some optimal solution. First we show that the graph induced by the nodes $v_1, v_2, \cdots, v_k$ is connected.

After removing v_1, the remaining problem is to remove $k-1$ nodes in $T\backslash\{v_1\}$. By induction, v_2 is a leaf of $T\backslash\{v_1\}$ and $v_2, v_3, \cdots, v_k$ are connected (or there is an equivalent solution with this property). Let us consider this other solution: $v_2, v_1, v_3, \cdots, v_k$. If v_2 is a leaf of T, then we are done. If v_2 is not a leaf of T and by induction v_2 is a leaf of $T\backslash\{v_1\}$, then v_1 and v_2 are neighbors. With this same argument, we can say that $v_1, v_2, \cdots, v_k$ are connected.

Denote the set $S = \{v_1, v_2, \cdots, v_k\}$. If any node in S is a leaf, then we are done. Consider every node in S is an interior node and let l_1 be the closest leaf to S. We show that for every path P of length $\geq L$ that disappears when $l_1, v_2, \cdots, v_k$ removed, there is a path P' of length $\geq L$ that disappears when $v_1, v_2, \cdots, v_k$ are removed and length $(P') \geq L$.

Let P be the path of length $\geq L$ that disappears when l_1 is removed. Note that P starts at l_1. If P contains some nodes in S, then $P' = P$. So assume P does not contain any node in $S = \{v_1, v_2, \cdots, v_k\}$, but it is destroyed when removing $l_1, v_2, \cdots, v_k$.

Let Q be the path from l_1 to S (see Figure 8.1). Let l_2 be the last node of P belonging to Q and l_3 be the last node of P (possibly $l_3 = l_2$). Let m be any leaf of T that can be reached starting from S without using any edge of Q. Let P' be the path from m to l_3. Clearly, P' disappears if some node in S is removed. We show that P' has length $\geq L$.

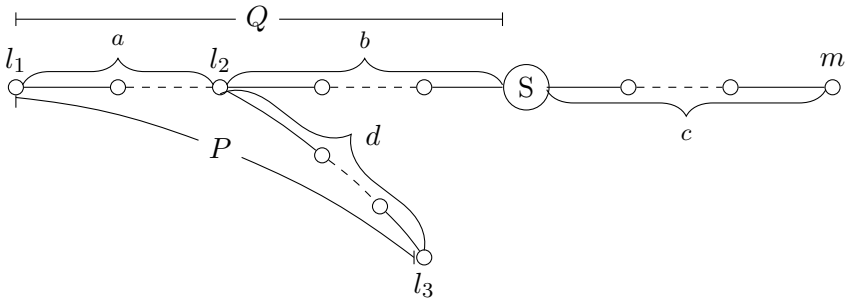

Figure 8.1: Illustration of the proof of Theorem 8.2.2.

Let the distance from l_1 to l_2 be a, l_2 to S be b, S to m be c and l_2 to l_3 be d (could be zero). Since m is a leaf and l_1 is the closest leaf to S, so

$$\text{dist}(S, m) \geq \text{dist}(S, l_1) \implies c \geq a + b$$

Now, length of $P' \geq c + b + d \geq a + b + b + d > L$ because $b > 0$ and $a + d =$ length of $P \geq L$. Clearly, $P_1 \neq P_2 \implies P_1' \neq P_2'$ because P_1 and P_2 have different final nodes. □

According to the preceding theorem, we must always remove a leaf from a tree at each step in order to maximize the path of length at least L, but we do not know which leaf should be eliminated. Furthermore, it is not true that at every stage iteration will choose the best leaf in order to achieve the best result and we call a leaf is the best if it destroys less connection of length at least L comparing to other leaves. Let us consider the tree depicted in Figure 8.2. Assume that our goal in this example is to maximize the number of node pairs connected by a path of length $L \geq 7$ that survive after removing at most $K = 2$ nodes with unit costs and node weights from the given tree T. We will destroy 4 paths of length $L \geq 7$ if we remove node 1, while we will destroy 3 paths of length $L \geq 7$ if we remove node 7. After deleting node 7, the best leaf to delete is node 6, because we would be destroying 2 paths of length $L \geq 7$. So, in total, $(3 + 2) = 5$ paths of length $L \geq 7$ will be destroyed by deleting nodes 7 and 6, respectively, where the best leaf will be removed at each step. However, removing node 1 and subsequently node 2 will destroy, in total, $(4 + 0) = 4$ paths of length $L \geq 7$. As a result, we will delete nodes 1 and 2 instead of nodes 7 and 6, despite the fact that node 1 was not the ideal leaf to remove at first step.

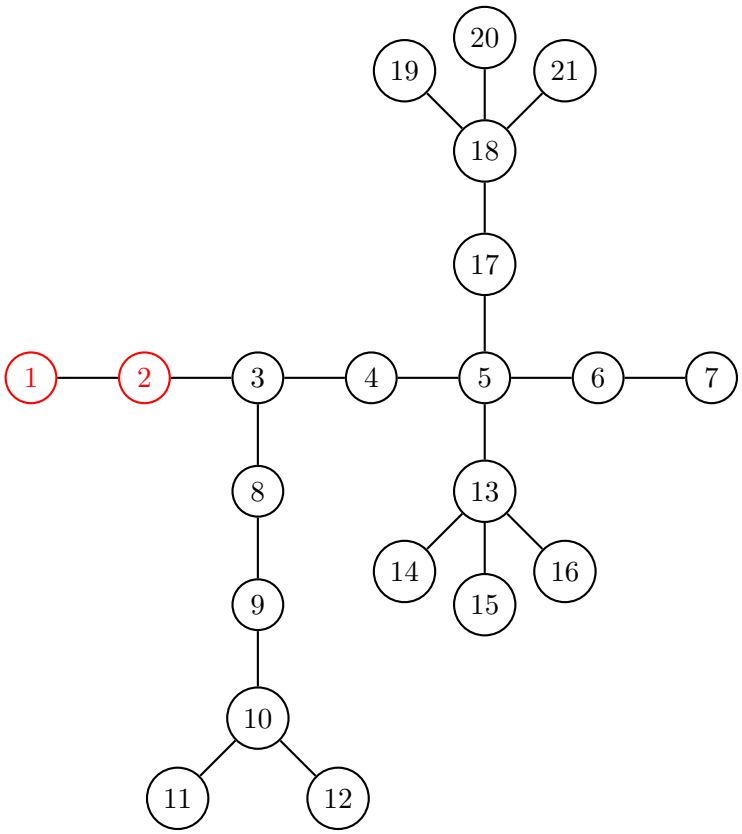

Figure 8.2: Solution for a D-CNDP of Class 2 when $L = 7$ and $K = 2$ with unit costs and unit node weights.

8.3 D-CEDP on trees for Class 1

The goal is to reduce the number of node-pairs remaining connected by a path of length at most L, for some natural number L, that survive in a tree $T(V, E)$ after at most K edges have been eliminated.

In order to solve the problem by dynamic programming, we define the following functions.

- $F_a(m_1, m_2, \cdots, m_{L-1}, k)$ = minimum number of paths of length $\leq L$ surviving in T_a when k edges are removed from T_a and $m_1, m_2, \cdots, m_{L-1}$ represents the number of nodes surviving in T_a having distance $1, 2, \cdots, (L-1)$ respectively from a.

- $G_{a_i}(m_1, m_2, \cdots, m_{L-1}, k)$ = minimum number of paths of length $\leq L$ surviving in the subtree $T_{a_{i,s}} = a + T_{a_i} + \cdots + T_{a_s}$ when k edges are removed from $T_{a_{i,s}}$ and $m_1, m_2, \cdots, m_{L-1}$ represents the number of nodes surviving in $T_{a_{i,s}}$ having original distance $1, 2, \cdots, (L-1)$ respectively from a.

We let the function values be infinity whenever the conditions cannot be satisfied. The values for F and G can be computed by traversing the tree in postorder (from leaves to root),

by means of the following relations:

$$F_a(m_1,\cdots,m_{L-1},k) = G_{a_1}(m_1,\cdots,m_{L-1},k) \quad \text{for any non-leaf node } a \in V; \tag{8.11}$$

$$G_{a_i}(m_1,\cdots,m_{L-1},k) = \begin{cases} \min\{F_{a_i}(m'_1,\cdots,m'_{L-1},0) + G_{a_{i+1}}(m''_1,\cdots,m''_{L-1},0) \\ \quad + \sum_{l=1}^{L}\sum_{j=0}^{l-1} m'_j m''_{l-1-j} : \\ \quad m'_j = 0,\ldots,\left|V(T_{a_{i,j}})\right| - 1,\ m''_j = m_j - m'_{j-1}\} & \text{if } k = 0, \\ \min\{\min\{F_{a_i}(m'_1,\cdots,m'_{L-1},q) + G_{a_{i+1}}(m_1,\cdots,m_{L-1},k-1-q) : \\ \quad m'_j = 0,\ldots,\left|V(T_{a_{i,j}})\right| - 1,\ q = 0,\ldots,k-1\}, \\ \quad \min\{F_{a_i}(m'_1,\cdots,m'_{L-1},q) + G_{a_{i+1}}(m''_1,\cdots,m''_{L-1},k-q) \\ \quad + \sum_{l=1}^{L}\sum_{j=0}^{l-1} m'_j m''_{l-1-j} : m'_j = 0,\ldots,\left|V(T_{a_{i,j}})\right| - 1, \\ \quad m''_j = m_j - m'_{j-1},\ q = 0,\ldots,k\}\} & \text{if } k > 0, \end{cases} \tag{8.12}$$

for any non-leaf node $a \in V$ where $i < s$ and $\left|V(T_{a_{i,j}})\right|$ denotes the number of nodes in T_{a_i} at distance j from a_i for each $j = 1,\ldots,L-1$.

The initial conditions on each leaf a and on each rightmost subtree T_{a_s} are the following:

$$F_a(m_1,\cdots,m_{L-1},k) = \begin{cases} 0 & \text{if } m_1 = \cdots = m_{L-1} = 0, k = 0, \\ \infty & \text{otherwise,} \end{cases} \tag{8.13}$$

$$G_{a_s}(m_1,\cdots,m_{L-1},k) = \begin{cases} \min\{F_{a_s}(m'_1,\cdots,m'_{L-1},k) + 1 + m'_1 + \cdots + m'_{L-1} : \\ \quad m'_j = m_{j+1}, j = 0,\ldots,L-1\} \quad \text{if } k = 0 \text{ or } m_j > 0 \text{ for some } j \in \{1,\ldots,L-1\}, \\ \min\{F_{a_s}(m'_1,\cdots,m'_{L-1},k-1) : \\ \quad m'_j = 0,\ldots,\left|V(T_{a_{s,j}})\right| - 1\} \quad \text{if } k > 0 \text{ and } m_j = 0 \text{ for all } j \in \{1,\ldots,L-1\}, \\ \infty \qquad \text{otherwise.} \end{cases} \tag{8.14}$$

Equation (8.11) follows because $T_a = T_{a_{1,s}}$ for any non-leaf node $a \in V$.

Recursion (8.12) can be interpreted as follows:

The case $k = 0$ means that we are not removing any edge from $T_{a_{i,s}} = T_{a_i} + T_{a_{i+1,s}}$. Since we have to keep everything, we are not allowed to remove anything from F and G. The expression $F_{a_i}(m'_1,\cdots,m'_{L-1},0)$ gives the minimum number of paths in T_{a_i} of length

$1,\ldots,L-1$ with the condition that m'_j nodes are connected at distance j from the root a_i and $G_{a_{i+1}}(m''_1,\cdots,m''_{L-1},0)$ gives the minimum number of paths in $T_{a_{i+1,s}}$ of length $1,\ldots,L-1$ under the condition that m''_j nodes are connected at distance j to the root a. Since we are not removing any edge, we have to add the paths of length $1,\ldots,L-1$ passing through the edge a to a_i, i.e., $\sum_{l=1}^{L}\sum_{j=0}^{l-1} m'_j m''_{l-1-j}$ paths. Hence by definition of F and G the minimum number of paths that survive in $T_{a_{i,s}}$ when we are not removing anything will be $G_{a_i}(m_1,\cdots,m_{L-1},0) = \min\{F_{a_i}(m'_1,\cdots,m'_{L-1},0) + G_{a_{i+1}}(m''_1,\cdots,m''_{L-1},0) + \sum_{l=1}^{L}\sum_{j=0}^{l-1} m'_j m''_{l-1-j}\}$.

The case $k>0$ means that we have to remove at least one edge. Note that the edge e (which connects a to a_i) is the connecting edge between the functions F and G. There are two cases to compute the value of $G_{a_i}(m_1,\cdots,m_{L-1},k)$ based on the edge e, either we remove e or we have to keep it. Consider first the case when the edge e is removed. In this case when q edges have been removed from T_{a_i}, exactly $k-1-q$ edges must be removed from $T_{a_{i+1,s}}$. Thus the minimum number of paths of length $1,\ldots,L-1$ at the respective distance to the root that survive in $T_{a_{i,s}}$ when q edges are removed from T_{a_i} (and the other $k-1-q$ edges are removed from $T_{a_{i+1,s}}$) is given by the formula $\min_q\{F_{a_i}(m'_1,\cdots,m'_{L-1},q) + G_{a_{i+1}}(m''_1,\cdots,m''_{L-1},k-1-q)\}$.

Consider now the case when the edge e is not removed. Expression $F_{a_i}(m'_1,\cdots,m'_{L-1},q)$ gives the minimum number of paths of length $1,\ldots,L-1$ and m'_j nodes are connected to the root a_i at distance j that survive in T_{a_i} when q edges are removed from T_{a_i}. Since q edges have been removed from T_{a_i}, exactly $k-q$ edges must be removed from $T_{a_{i+1,s}}$. The minimum number of paths of length $1,\ldots,L-1$ at distance j from the root a that survive in $T_{a_{i+1,s}}$ when $k-q$ edges are removed from $T_{a_{i+1,s}}$ is given by $G_{a_{i+1}}(m''_1,\cdots,m''_{L-1},k-q)$. Thus the expression $F_{a_i}(m'_1,\cdots,m'_{L-1},q) + G_{a_{i+1}}(m''_1,\cdots,m''_{L-1},k-q)$ gives the minimum number of paths of length $1,\ldots,L-1$ that survive in T_{a_i} or $T_{a_{i+1,s}}$ when q edges are removed from T_{a_i} (and the other $k-q$ edges are removed from $T_{a_{i+1,s}}$). Since we are not removing the edge e, we have to add the paths of length $1,\ldots,L-1$ connecting nodes of T_{a_i} to nodes of $T_{a_{i+1,s}}$, i.e., $\sum_{l=1}^{L}\sum_{j=0}^{l-1} m'_j m''_{l-1-j}$ paths. By taking the minimum over $q=0,\ldots,k$, we find the value of $G_{a_i}(m_1,\cdots,m_{L-1},k)$.

Equation (8.13) handles the case of a one-node tree. Since $a\in V$ is a leaf, it is not possible to remove any edge ($k=0$) and only a is connected to itself ($m_1=\cdots=m_{L-1}=0$) and the number of paths surviving in T_a is 0.

For a justification of (8.14), recall that $T_{a_{s,s}} = a + T_{a_s}$. Here the distinction is based on whether we keep the the edge a to a_s or not. We must keep the edge a to a_s exactly when $k=0$ or at least one of the parameters m_j's is positive. While if $k>0$ and all m_j's are zero, then we are forced to remove the edge a to a_s.

Assuming that the tree T is rooted at node 1, the optimal value for the problem is given

by

$$\text{OPT} = \min\{F_1(m_1, \cdots, m_{L-1}, k) : \sum_{j=1}^{L-1} m_j < n,\ k = 0, \cdots, K\}, \tag{8.15}$$

and the optimal solution is recovered by backtracking. We get the following proposition.

Proposition 8.3.1. *D-CEDP on trees for Class 1 with unit costs and unit edge weights can be solved by recursion* (8.11)–(8.14) *in* $\mathcal{O}(n^{3L-2}K^2)$ *time.*

Proof. For each node $a \in V$ there are at most $\mathcal{O}(n^L)$ values for $m_0, m_1, \cdots, m_{L-1}$ and $\mathcal{O}(K)$ values for k; this gives $\mathcal{O}(n^{2L-1}K)$ values of F and G to compute. The heaviest computation is that of equation (8.12) that requires at most $\mathcal{O}(n^{L-1}K)$ steps. Hence in the worst case a number of operations bounded by $\mathcal{O}(n^{3L-2}K^2)$ are required. □

We point out that the proposed dynamic program is more of a theoretical contribution to the class of distance functions, as the algorithm becomes intractable even for modest values of parameter L in practice.

8.4 IP formulations for D-CEDP

Let $G(V, E)$ be an undirected graph with $|V| = n$ nodes and let $K > 0$ be an integer which is the maximum number of edges that can be removed. Now we create the graph $G' = (V', E')$ by subdividing every edge of G as described in Chapter 4. Now we consider the restricted CNDP on G' with the allowed set $A = V' \setminus V$, i.e., the set of newly created vertices. Then the problem of removing at most K edges on G becomes the problem of removing at most K nodes on G' but here we are allowed to remove K nodes only from the set A of newly created nodes. Our objective is to minimize the number of node-pairs still connected by a path of length $\leq L$, for some given natural number L, surviving in a graph $G(V, E)$ after having removed at most K nodes from A.

The optimal solution of the D-CEDP for minimizing the number of paths of length $\leq L$ can be determined by using an integer programming (IP) formulation. The IP formulation is described below.

Define the binary variables u_{ij}^l and v_i as follows:

$$u_{ij}^l = \begin{cases} 1, & \text{if } i, j \text{ are connected by a path of length} \leq l \\ 0, & \text{otherwise,} \end{cases}$$

where $i, j \in V'$ and $l \in \{1, 2, ..., 2L\}$,

and

$$v_i = \begin{cases} 1, & \text{if node } i \text{ is deleted from the graph} \\ 0, & \text{otherwise.} \end{cases}$$

Then the problem admits the following integer programming formulations:

$$\text{(D-CEDP)} \qquad minimize \sum_{i,j \in V' \setminus V,\ i<j} u_{ij}^{2L} \tag{8.16}$$

subject to

$$u_{ij}^{1} + v_i + v_j \geq 1, \forall ij \in E' \tag{8.17}$$

$$u_{ij}^{l} + u_{jk}^{1} - u_{ki}^{l+1} \leq 1,\ \forall\ i \in V',\ \forall\ jk \in E',\ j < k,\ \forall\ l \in \{1, 2, ..., 2L-1\} \tag{8.18}$$

$$u_{ik}^{l} + u_{kj}^{1} - u_{ji}^{l+1} \leq 1,\ \forall\ i \in V',\ \forall\ jk \in E',\ j < k,\ \forall\ l \in \{1, 2, ..., 2L-1\} \tag{8.19}$$

$$u_{ij}^{l} \leq u_{ij}^{l+1},\ i < j,\ \forall i, j \in V',\ \forall\ l \in \{1, 2, ..., 2L-1\} \tag{8.20}$$

$$\sum_{i \in A} v_i \leq K \tag{8.21}$$

$$v_i = 0,\ \forall i \notin A, \tag{8.22}$$

$$u_{ij}^{l} \in \{0, 1\},\ \forall i, j \in V',\ \forall\ l \in \{1, 2, ..., 2L\} \tag{8.23}$$

$$v_i \in \{0, 1\},\ \forall i \in V'. \tag{8.24}$$

The objective of this model is to find a set S of K nodes from the set A of length $\leq L$ whose removal cause minimum pairwise connectivity in the induced subgraph $G(V \setminus S)$. Since we are working on G', the graph has double length and the objective function (8.16) minimizes the number of node-pairs still connected by a path of length $\leq 2L$ surviving in a graph G' after having removed at most K nodes from A. Constraint (8.17) means that if nodes i and j are in different components and there is an edge between them, then one of them should be deleted. As shown in [60], constraints (8.18) and (8.19) altogether indicate that if node i is connected to node j by a path of length l and node j is connected to node k by an edge, then node i must also be connected with node k by a path of length $l+1$. Constraint (8.20) implies that if nodes i and j are connected by a path of length l, then they are also connected by a path of length $l+1$. Constraint (8.21) guarantees that the number of nodes to be deleted is at most K. Constraint (8.22) says that only the allowed vertices can be removed. Finally, (8.23) and (8.24) define the proper domains for the variables used.

Chapter 9

Conclusion

This chapter summarizes the contribution of the thesis and discusses avenues for possible future research.

9.1 Summary of contribution

In this thesis we focused our study on identifying critical nodes and/or edges from a graph, whose deletion results in the minimum pairwise connectivity among the remaining nodes. We first studied a particular case of the critical node detection problem, namely the Distance-based Critical Node Detection Problem (D-CNDP) over paths. We considered four versions of the D-CNDP on paths and provided closed-form solution to solve the problems.

We then investigated the critical edge detection problem over trees. We showed that the cases with unit connection costs and unit or arbitrary edge weights are solvable in polynomial time through dynamic programming approaches. We also showed that CNEDP-1 (i.e., a joint weight limit for the removal of nodes and edges is given) can be solved in polynomial time when the connection costs are all unitary and the node and edge weights are general nonnegative numbers, while CNEDP-2 (i.e., two separate weight limits for the removal of nodes and edges are given) can be solved in polynomial time when the connection costs are fixed to 1 as well as the node and edge weights. We proved that CNEDP-2 with unit connection costs and general nonnegative weights is $\mathcal{NP}$-complete even on a path through a reduction from Partition.

The CNDP is known to be $\mathcal{NP}$-hard on general graphs [8]. In [27], it has been shown that the pairwise connectivity CNDP on a tree is $\mathcal{NP}$-hard even if the node weights are all equal to 1 and the connection costs are 0/1. On the one hand we proved that CEDP, CNEDP-1, and CNEDP-2 are all $\mathcal{NP}$-hard under the assumptions that the node/edge weights are unitary and the connection costs are 0/1, and on the other hand we have provided polynomial time algorithms when the connection costs are square 0/1 connection costs. The problem was then analyzed when the number of leaves of the tree is a constant. We proposed dynamic programming algorithms for CNDP, CEDP, and CNEDP-1 over trees (including the special case of

a path) with arbitrary 0/1 connection costs and general nonnegative node/edge weights. We also presented integer programming formulations which provide optimal solutions for CNDP, CEDP, CNEDP-1, and CNEDP-2.

Finally we proposed dynamic programming algorithms for the Distance-based Critical Node/Edge Detection Problem over trees. We have found that the D-CNDP based on Class 1 and Class 2 can be solved polynomially when its input parameter L is considered fixed. We proved that to remove K nodes from a tree for maximizing the number of paths of length at least L, there is an optimal solution by recursively removing a leaf. We also provided polynomial time algorithms to solve the D-CEDP on trees of Class 1 when parameter L is fixed. We presented integer programming formulations for the D-CEDP.

9.2 Directions for future work

We wrap up this dissertation by addressing some unresolved questions.

We have seen that all most all of the variants of the CNDP, CEDP, CNEDP-1, and CNEDP-2 considered in this thesis can be solved in polynomial time on trees. In the future, it would be interesting to see extensions of our results to more general graphs rather than trees.

We have seen in Chapter 6 that with a constant number of leaves our problems are polynomial time solvable, but these problems are not fixed-parameter tractable. A problem is said to be a fixed-parameter tractable (FPT) if it can be solved by algorithms that are exponential only in the size of a fixed parameter, while polynomial in the size of the input. In this context one natural question raises: Does FPT algorithm exist for the problems that we considered or it is impossible to find it?

As shown in Chapter 8, since all the versions of the distance-based critical node/edge problem over trees that we considered are polynomial time solvable when parameter L is constant, it would be interesting to see what happens if the parameter L is not constant.

Finally, there are no computational experiments in this dissertation. This is also an aspect that should be explored.

Bibliography

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

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

[3] Albert, R., Jeong, H., A. L. Barabasi, A. L.: *Error and attack tolerance of complex networks*, Nature, 406, 378–382 (2000).

[4] Alozie, G.U., Arulselvan, A., Akartunalı, K., Pasiliao Jr, E.L.: *Efficient methods for the distance-based critical node detection problem in complex networks*, Computers & Operations Research, 131(1):105254 (2021).

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

[6] Aringhieri, R., Grosso, A., Hosteins, P., Scatamacchia, R.: *Local search metaheuristics for the critical node problem*, Networks, 67(3), 209–221 (2016).

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

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

[9] Arulselvan, A., Commander, C. W., Pardalos, P. M. and Shylo, O.: *Managing network risk via critical node identification*, Gulpinar, N., Rustem, B. (eds.) Risk Management in Telecommunication Networks, 2010.

[10] Arulselvan, A., Commander, C.W., Shylo, O., Pardalos, P.M.: *Cardinality constrained critical node detection problem, in: Performance Models and Risk Management in Communications Systems*, Springer, 79–91 (2011).

[11] Aspnes, J., Chang, K., Yampolskiy, A.: *Inoculation strategies for victims of viruses and the sum-of-squares partition problem*, in: Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms, Society for Industrial and Applied Mathematics, 43–52 (2005).

[12] Avidor, A., Langberg, M.: *The multi-multiway cut problem*, Theoret. Comput. Sci., 377 (1–3), 35–42 (2007).

[13] Berger, A., Grigoriev, A., van der Zwaan, R.: *Complexity and approximability of the k-way vertex cut*, Networks, 63(2), 170–178 (2014).

[14] Boginski, V., Commander, C.: *Identifying critical nodes in protein-protein interaction networks*, In: S. Butenko, W. Chaovalitwongse, P. Pardalos (eds.) Clustering challenges in biological networks, 153–167. World Scientific (2009).

[15] Borgatti, S.P.: *Identifying sets of key players in a network*, Computational and Mathematical Organization Theory, 12, 21–34 (2006).

[16] Brown, G., Carlyle, M., Salmerón, J., Wood, K.: *Defending critical infrastructure*, Interfaces, 36(6), 530–544 (2006).

[17] Buluc, A., Meyerhenke, H., Safro, I., Sanders, P., Schulz, C.: *Recent advances in graph partitioning*, in: Algorithm Engineering, Springer, 117–158 (2016).

[18] Chopra, S., Rao, M.R.: *On the multiway cut polyhedron*, Networks, 21 (1), 51–89 (1991).

[19] Chopra, S., Owen, J.H.: *Extended formulations for the A-cut problem*, Math. Program., 73 (1), 7–30 (1996).

[20] Church, R.L., Scaparra, M.P., Middleton, R.S.: *Identifying critical infrastructure: The median and covering facility interdiction problems*, Ann. Assoc. Am. Geogr. 94(3), 491–502 (2004).

[21] Cohen, R., Ben Avraham, D., Havlin, S.: *Efficient immunization strategies for computer networks and populations*, Physical Review Letters, 91, 247901–247905 (2003).

[22] Commander, C.W., Pardalos, P.M., Ryabchenko, V., Uryasev, S., Zrazhevsky, G.: *The wireless network jamming problem*, Journal of Combinatorial Optimization, 14, 481–498 (2007).

[23] Corley, H., Sha, D.Y.: *Most vital links and nodes in weighted networks*, Oper. Res. Lett., 1(4), 157–160 (1982).

[24] Costa, M.-C., Letocart, L., Roupin, F.: *Minimal multicut and maximal integer multiflow: A survey*, European J. Oper. Res., 162 (1), 55–69 (2005).

[25] Crucitti, P., Latora, V., Marchiori, M., Rapisarda, A.: *Efficiency of scale-free networks: Error and attack tolerance*, PHYSICA A, 320, 622–642 (2003).

[26] Dahlhaus, E., Johnson, D.S., Papadimitriou, C.H., Seymour, P.D., Yannakakis, M.: *The complexity of multiterminal cuts*, SIAM J. Comput., 23 (4), 864–894 (1994).

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

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

[29] Dinh, T., Xuan, Y., Thai M.T. Park, E., Znati, T.: *On approximation of new optimization methods for assessing network vulnerability*, In: Proceedings of INFOCOM, IEEE (2010).

[30] Dinh, T.N., Thai, M.T.: *Precise structural vulnerability assessment via mathematical programming*, in: Military Communications Conference, MILCOM 2011, IEEE, 1351–1356 (2011).

[31] Dinh, T.N., Xuan, Y., Thai, M.T., Pardalos, P.M., Znati, T.: *On new approaches of assessing network vulnerability: Hardness and approximation*, Networking, IEEE/ACM Transactions on, 20(2), 609–619 (2012).

[32] Dinh, T.N., Thai, M.T.: *Assessing attack vulnerability in networks with uncertainty*, in: IEEE International Conference on Computer Communications (INFOCOM), IEEE, 2015.

[33] Elefteriadou L.: *Highway capacity, in Kutz M. (ed.) Handbook of transportation engineering*, New York, McGraw-Hill, chapter 8 (2004).

[34] Fan, N., Pardalos, P.: *Robust optimization of graph partitioning and critical node detection in analyzing networks*, In: W. Wu, O. Daescu (eds.) International Conference on Combinatorial Optimization and Applications, Lecture Notes in Computer Science, vol. 6508, pp. 170–183. Springer (2010).

[35] Fjallstrom, P.-O.: *Algorithms for Graph Partitioning: A Survey*, Linkoping University Electronic Press Linkoping, Vol. 3, (1998).

[36] Garey, M., Johnson, D.: *Computers and intractability*, W. H. Freeman and Co., San Francisco, Calif. (1979), A guide to the theory of NP-completeness, A Series of Books in the Mathematical Sciences.

[37] Garg N., Vazirani V.V., Yannakakis M.: *Primal-dual approximation algorithms for integral flow and multicut in trees*, Algorithmica, 18, 3–30 (1997).

[38] Goldschmidt, O., Hochbaum, D.S.: *A polynomial algorithm for the k-cut problem for fixed k*, Math. Oper. Res. 19 (1), 24–37 (1994).

[39] Grubesic, T.H., Matisziw T.C., Murray, A.T., Snediker D.: *Comparative approaches for assessing network vulnerability*, Int. Reg. Sci. Rev., 31(1), 88–112 (2008).

[40] Guo, J., Niedermeier, R.: *Fixed-parameter tractability and data reduction for multicut in trees*, Networks, 46 (3), 124–135 (2005).

[41] He, X.: *An improved algorithm for the planar 3-cut problem*, J. Algorithms, 12 (1), 23–37 (1991).

[42] He, J., Liang, H., Yuan, H.: *Controlling infection by blocking nodes and links simultaneously*, In: N. Chen, E. Elkind, E. Koutsoupias (eds.) International Workshop on Internet and Network Economics (WINE 2011), Lecture Notes in Computer Science, vol. 7079, pp. 206–217. Springer (2011).

[43] Hooshmand, F., Mirarabrazi,F., MirHassani, S.A.: *Efficient Benders decomposition for distance-based critical node detection problem*, Omega, 93:102037, (2020).

[44] Houck, D.J., Kim, E., O'Reilly, G.P., Picklesimer, D.D., Uzunalioglu, H.: *A network survivability model for critical national infrastructures*, Bell Labs Technical J., 8(4), 153–172 (2004).

[45] Israeli, E., Wood, R.K.: *Shortest-path network interdiction*, Networks, 40(2), 97–111 (2002).

[46] Jenelius, E., Petersen, T., Mattsson, L.G.: *Importance and exposure in road network vulnerability analysis*, Transportation Research Part A: Policy and Practice 40(7), 537–560 (2006).

[47] Jhoti, H., Leach, A.R.: *Structure-based Drug Discovery*, Springer, 2007.

[48] Krebs V.: *Uncloaking terrorist networks*, First Monday, 7 (2002).

[49] Kuhlman, C., Kumar, V., Marathe, M., Ravi, S., Rosenkrantz, D.: *Finding critical nodes for inhibiting diffusion of complex contagions in social networks*. In: J. Balc´azar, F. Bonchi, A. Gionis, M. Sebag (eds.) Joint European Conference on Machine Learning and Knowledge Discovery in Databases, Lecture Notes in Computer Science, vol. 6322, pp. 111–127. Springer (2010).

[50] Kuhlman, C., Tuli, G., Swarup, S., Marathe, M., Ravi, S.: *Blocking simple and complex contagion by edge removal*, In: 13th International Conference on Data Mining, pp. 399–408. IEEE (2013).

[51] Lalou, M., Tahraoui, M., Kheddouci, H.: *Component-cardinality-constrained critical node problem in graphs*, Discrete Applied Mathematics, 210, 150–163 (2016).

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

[53] Liljefors, T., Krogsgaard-Larsen, P., Madsen, U.: *Textbook of Drug Design and Discovery*, CRC Press, 2002.

[54] Marx, D.: *Parameterized graph separation problems*, Theoret. Comput. Sci., 351 (3), 394–406 (2006).

[55] Matisziw, T.C., Murray, A.T.: *Modeling s-t path availability to support disaster vulnerability assessment of network infrastructure*, Comput. Oper. Res., 36, 16–26 (2009).

[56] Miller, J. C., and Hyman, J. M.: *Effective vaccination strategies for realistic social networks*, Physica A: Statistical Mechanics and its Applications, 386(2):780–785, (2007).

[57] Myung, Y.-S. and Kim, H.-J.: *A cutting plane algorithm for computing k-edge survivability of a network*, European Journal of Operational Research, 156(3), 579–589 (2004).

[58] Nguyen, D., Shen, Y., Thai, M.: *Detecting critical nodes in interdependent power networks for vulnerability assessment*, IEEE Transactions on Smart Grid, 4(1), 151–159 (2013).

[59] Oosten, M., Rutten, J.H.G.C., Spieksma, F.C.R.: *Disconnecting graphs by removing vertices: a polyhedral approach*, Statistica Neerlandica 61(1), 35–60 (2007).

[60] Pavlikov, K.: *Improved Formulations for Minimum Connectivity Interdiction Problems*, Computers & Operations Research 97, 48-57 (2018).

[61] Pawson, T., Nash, P.: *Protein–protein interactions define specificity in signal transduction*, Genes Dev. 14(9), 1027–1047 (2000).

[62] Pothen, A.: *Graph partitioning algorithms with applications to scientific computing*, in: Parallel Numerical Algorithms, Springer, 323–368 (1997).

[63] Prieto, C., De Las Rivas, J.: *APID: agile protein interaction DataAnalyzer*, Nucleic Acids Res. 34 (suppl. 2), W298–W302 (2006).

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

[65] Purevsuren, D., Cui, G., Qu, M., Win, N.: *Hybridization of grasp with exterior path relinking for identifying critical nodes in graphs*, IAENG International Journal of Computer Science, 44(2) (2017).

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

[67] Salemi, H., Buchanan, A.: *Solving the distance-based critical node problem*, Optimization Online, (2020).

[68] Salmeron, J., Wood, K.R., Baldick, R.: *Analysis of electric grid security under terrorist threat*, IEEE Transactions on Power Systems, 19(2), 905–912 (2004).

[69] Scoglio, C., Schumm, W., Schumm, P., Easton, T., Roy Chowdhury, S., Sydney, A. and Youssef, M.: *Efficient Mitigation Strategies for Epidemics in Rural Regions*, PLoS ONE, 5(7): (2010).

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

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

[72] Shen, Y., Di, L., Wu, L., Yu, G., Tang, H., Yu, G.: *Hidden Markov models for corn progress percents estimation in multivariate time series*, In: International Conference on Agro-Geoinformatics (Agro-Geoinformatics) (2012).

[73] Shen, Y., Nguyen N.P. Xuan, Y., Thai, M.: *On the discovery of critical links and nodes for assessing network vulnerability*, IEEE/ACM Transactions on Networking, 21(3), 963–973 (2013).

[74] Smith, J. C. and Lim, C.: *Algorithms for discrete and continuous multicommodity flow network interdiction problems*, IIE Transactions, 39, 15–26 (2007).

[75] Tomaino, V., Arulselvan, A., Veltri, P., Pardalos, P.: *Studying connectivity properties in human protein-protein interaction network in cancer pathway*, In: P. Pardalos, P. Xanthopoulos, M. Zervakis (eds.) Data Mining for Biomarker Discovery, pp. 187–197, Springer (2012).

[76] Vazirani, V.: *Approximation Algorithms*, Springer Science & Business Media (2013).

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

[78] Ventresca, M., Aleman, D.: *A fast greedy algorithm for the critical node detection problem*, in: Combinatorial Optimization and Applications, Springer, 603–612 (2014).

[79] Ventresca, M., Aleman, D.: *A derandomized approximation algorithm for the critical node detection problem*, Comput. Oper. Res., 43, 261–270 (2014).

[80] Ventresca, M., Aleman, D.: *A region growing algorithm for detecting critical nodes*, in: Combinatorial Optimization and Applications, Springer, 593–602 (2014).

[81] Ventresca, M., Aleman, D.: *A randomized algorithm with local search for containment of pandemic disease spread*, Comput. Oper. Res., 48, 11–19 (2014).

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

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

[84] Westermarck, J., Ivaska, J., Corthals, G.L.: *Identification of protein interactions involved in cellular signaling*, Mol. Cell. Proteomics 12 (7), 1752–1763 (2013).

[85] Wollmer, R.: *Removing arcs from a network*, Operations Research, 12, 934–940 (1964).

[86] Wood, R.K.: *Deterministic network interdiction*, Mathematical and Computer Modelling, 17, 1–18 (1993).

[87] Yan, C., Wu, F., Jernigan, R.L., Dobbs, D., Honavar, V.: *Characterization of protein–protein interfaces*, Protein J. 27 (1), 59–70 (2008).

[88] Zhao, K., Kumar, A., Harrison, T. P. and Yen, J.: *Analyzing the resilience of complex supply network topologies against random and targeted disruptions*, IEEE Systems Journal, 5(1), 28–39 (2011).

[89] Zhou T., Fu Z.-Q., Wang B.-H.: *Epidemic dynamics on complex networks*, Progress in Natural Science, 16, 452–457 (2006).