Detecting Critical Node Structures on Graphs: A Mathematical Programming Approach

Jose L. Walteros*
Department of Industrial and Systems Engineering
University at Buffalo, 413 Bell Hall, Buffalo, New York, 14260
josewalt@buffalo.edu

Alexander Veremyev
Industrial Engineering and Management Systems Department
University of Central Florida, 12800 Pegasus Dr, Orlando, FL 32816-2993
alexander.veremyev@ucf.edu

Panos M. Pardalos
Industrial and Systems Engineering Department
University of Florida, 303 Weil Hall, Gainesville, Florida 32611,
pardalos@ufl.edu

Eduardo L. Pasiliao
Air Force Research Laboratory,
AFRL Munitions Directorate, Building 13 Eglin AFB, Florida 32542,
pasiliao@eglin.af.mil

Abstract

We consider the problem of detecting a collection of critical node structures of a graph whose deletion results in the maximum deterioration of the graph's connectivity. The proposed approach is aimed to generalize other existing models whose scope is restricted to removing individual and unrelated nodes. We consider two common metrics to quantify the connectivity of the residual graph: the total number of connected node pairs and the size of the largest connected component. We first discuss the computational complexity of the problem and then introduce a general mixed-integer linear formulation, which, depending on the kind of node structures, may have an exponentially large number of variables and constraints. To solve this potentially large model, we develop a branch-price-and-cut framework, along with some valid inequalities and preprocessing algorithms, in order to strengthen the formulation and reduce the overall execution time. We use the proposed approach to solve the problem for the cases where the node structures form cliques or stars and provide further directions on how to extend the framework for detecting other kinds of critical structures as well. Finally, we test the quality of our approach by solving a collection of real-life and randomly generated instances with various configurations, analyze the benefits of our model, and propose further enhancements.

Keywords critical node problem; network interdiction; combinatorial optimization; branch-price-and-cut; graph partitioning; mixed-integer programming.

*Corresponding author. Phone: 716-645-8876; Fax: 716-645-3302; Address: 413 Bell Hall, Buffalo, NY 14260; Email: josewalt@buffalo.edu

1 Introduction and Motivation

Consider a simple, non-empty, graph $G = (V, E)$, where V is a set of n nodes and $E \subseteq \binom{V}{2}$ is a set of m edges[1]. We will refer to each edge by either its index $e = 1, \ldots, m$ or by its pair of endpoints $\{i, j\}$. We say that two nodes i and j are adjacent if $\{i, j\} \in E$. For any node $i \in V$, let $\mathcal{N}_G(i)$ denote the set of nodes adjacent to i in G—i.e., the neighborhood of i in G. We say that H is a subgraph of G, if the nodes and edges of H, referred to as $V(H)$ and $E(H)$, are subsets of V and E, respectively. For any subset of nodes $N \subseteq V$, let $\mathcal{E}(N) = E \cap \binom{N}{2}$ be the set of edges so that, for each edge $e \in \mathcal{E}(N)$, both endpoints of e belong to N. Given a subset of nodes $N \subseteq V$, let $G[N]$ denote the subgraph induced by N. That is, $G[N]$ is the graph with node set N and edge set $\mathcal{E}(N)$. For any pair of nodes $\{i, j\} \in \binom{V}{2}$, we say that i and j are connected in G if there exists at least one path that connects i and j in G. Furthermore, we say that N induces a connected component in G if every pair of nodes in N are connected in G and every node in $V \setminus N$ is disconnected from all nodes in N. The collection of all connected components of G is denoted Θ_G and the number of connected node pairs of a connected component $H \in \Theta_G$ is then $\binom{|V(H)|}{2}$.

We use the term node structure when referring to a subgraph that has a generic predefined set of characteristics that are common to other subgraphs, or if it belongs to a certain collection of predefined subgraphs. For instance, cliques and stars are two typical types of node structures. In this paper, we say a clique[2] K is a subgraph of G in which every two nodes in $V(K)$ are adjacent in G—i.e., $\binom{V(K)}{2} \subseteq E$; a star S is a subgraph of G given by a hub node denoted $h(S)$, a set of leaf nodes $l(S) \subseteq \mathcal{N}_G(h(S))$, and the set of edges connecting the hub with the leaves. A clique K is said to be maximal if it is not a subset of any other clique in G. Similarly, a star S is maximal if $l(S) = \mathcal{N}_G(h(S))$ (i.e., it comprises the hub $h(S)$ and all of its neighbors). A star is also said to be induced if its leaf nodes are non-adjacent in G. The size of a clique is equal to the number of its nodes and the size of a star is given by the number of its leaves. Thus, we will say that a single node is a clique of size one or a star of size zero.

In this study, we consider the problem of optimally deteriorating two connectivity properties of a graph G—i.e., the number of connected node pairs and the size of the largest component—by removing some set of critical node structures from a predefined collection, subject to a budgetary constraint.

Over the last few years there has been an increasing interest in developing models that study the vulnerability of graphs with respect to possible failures of some of its elements—i.e., node and edge subsets. These models have grown in popularity because of their ability to analyze the negative effects on a graph in case some of its elements become impaired or destroyed. Since the failure of an element may be caused by different actors—e.g., adversarial attacks, random failures, or natural disasters—most of these problems are often solved for designing optimal attacks that maximize the damage inflicted to the graph, for devising defensive strategies that efficiently mitigate the effects of any possible disruption, or for analyzing the vulnerability of underlying network structures.

Several studies regarding these problems and similar variations can be found in the literature, as well as many applications in different areas—e.g., homeland security [32, 37], evacuation planning [49], immunization strategies [67], energy systems [61], and transportation networks [39]. However, despite the broad interest on the subject, there seems to be no overall agreement on a base name

[1]For a given set N, we use $\binom{N}{2}$ to denote the set of all possible unordered pairs created with elements from N. That is, $\binom{V}{2} = \{\{i, j\} | i, j \in N, i \neq j\}$. Similarly, $\binom{V}{3} = \{\{i, j, k\} | i, j, k \in N, i \neq j \neq k\}$ generalizes the case for triplets.

[2]In the literature, cliques are often associated with the set of nodes rather than with the subgraphs induced by them. Given the fact that node structures are defined here as subgraphs, we will use both the subset or nodes or the induced subgraph when referring to a clique.

that encompasses and describes the general case. Most authors often use the names: critical elements problems [6], network interdiction problems [38], node (edge) deletion problems [55], most-vital elements problems [10], key players identification [14], blocker problems [46], and disruptor problems [23, 65], among others.

In many of these studies, a wide variety of structural properties and their corresponding measures have been proposed to quantify the disruption inflicted on the graph G. In general, these measures are either associated with: (1) optimal solutions to flow problems over G, such as shortest paths, maximum flow, or minimum cost flow problems [17, 19, 33, 38, 41, 44, 49, 78, 79]; (2) the sizes or relative weight of some topological node or edge structures in G, like spanning trees [28], dominating sets [47], central nodes—e.g., the 1-median or 1-center nodes [10], matchings [80, 81], independent sets [9], cliques [46] and node covers [9]; and (3) connectivity and cohesiveness properties of G, such as, the total number of connected node pairs [1, 6, 21, 22, 23, 51, 52, 65, 69, 70, 72, 73], the weight of the connections between the node pairs [6, 21, 73], the size of the largest component [13, 30, 55, 63, 64, 73], the total number of components [5, 63, 64, 65, 68], distance based connectivity metrics [74], and the graph information entropy [14, 35, 56, 73].

While, the terms *interdiction* [38, 79], *blockers* [47], and *most vital elements* [10] have been used when analyzing various graph properties, in this context, the term *critical elements*—and *criticality* in general—appears to be mostly adopted as a reference to problems that aim to deteriorate the connectivity of the given graph (see the recent studies [42, 73] and the references therein). Consequently, as we attempt to identify node structures in a graph whose presence is *critical* for maintaining the graph's connectivity, we refer to the collective of problems we approach in this paper as *critical node structure detection problems.* We will resort however to the more specific name—e.g., critical cliques or critical stars—when referring to a particular case of node structures.

In addition to the measures used to analyze the disruption inflicted on the graph's connectivity, other important factors that must be considered when dealing with this kind of problems are the characteristics that the critical elements are required to possess. With very few exceptions [30, 75], most of the literature deals with the removal of independent and unrelated node (edge) sets, without taking any further consideration about the structure of such nodes (edges). In contrast, in many real-life applications, including social networks analysis, immunization strategies, and communications, there exist some problems that can be modeled by having the critical nodes (edges) forming particular node structures, such as cliques, stars, paths, connected subgraphs, etc.

Take for instance the case in which there is an interest in analyzing the cascading effects that may occur whenever some nodes fail. When a node gets destroyed and there is no immediate way of counteracting the source that inflicted the damage, it is possible that several (if not all) of the node's neighbors get affected as well. This kind of situations can be modeled by identifying *critical stars* rather than only critical nodes. Similarly, when studying a viral propagation over a graph, if several members within a dense subgraph—e.g., a clique—get "infected", because of the reinforcement effect that such members inflict over the subgraph, other members may become more prone to acquire the infection as well. Thus, identifying dense structures such as *critical cliques* may be useful for analyzing problems with strong propagation dynamics, like the dissemination of a marketing campaign over a social network or the contagion dynamics of a disease.

Moreover, nodes in many complex networks may be part of groups representing certain entities whose importance need to be evaluated. For example in a social network context, one may think of an organization, in which employees form teams or tight communities working on different projects [26]. In biological systems represented by protein-protein interaction networks, proteins form protein complexes or functional modules [66]. Protein complexes act as single multimolecular machines while functional modules consist of proteins participating in a particular cellular process. In addition, it has been documented that many real-life networks consist of motifs (patterns of interconnections) at numbers that are significantly higher than those in randomized networks [50]. Such motifs can be viewed as elementary building blocks of most complex networks. Hence, finding a set of critical groups of nodes which represent teams in an organization, protein complexes or functional modules, motifs, etc., in these contexts will shed light not only on a structural organization of complex networks, but also indicate which node structures play an important role in maintaining network integrity or act as intermediaries connecting different parts of networks.

For the particular case of identifying critical cliques, [75] proposed a two-stage heuristic decomposition strategy that first identifies a candidate clique partition and then uses this partition to reformulate and solve the problem as a generalized critical node problem (GCNP). To generate candidate clique partitions the authors tested two heuristic approaches and solve the resulting (GCNP) using a commercial optimizer. The main disadvantage of this approach is that the quality of the results depends drastically on how good the clique partitions that are used to reformulate the problem are and for the two algorithms the authors used, the results obtained by the reformulation can be quite erratic.

In this paper, we are interested in closing this gap by introducing a generic mathematical framework that can be used for identifying critical node structures in general. Our contributions are summarized as follows: (1) We present the problem of identifying a set of node structures (i.e., the critical node structures) in an undirected graph, so that after deleting the nodes and edges of such structures, subject to budgetary constraints, the graph gets maximally disconnected. We provide specific examples of how to use the proposed approach for the cases where the structures form stars or cliques. Nonetheless, our approach is general enough to be used for identifying other types of node structures as well. (2) We provide an analysis of the computational complexity of this problem for different types of node structures. (3) We develop a mathematical framework for analyzing the deterioration of two connectivity properties, the size of the largest component and the total number of connected node pairs. Depending on the context and the kind of structural property that is used, it is possible that the total number of variables and/or constraints required to model this problem grows exponentially with the size of the graph. To tackle this issue, we develop a branch-price-and-cut approach, which includes a particular branching strategy, a generic pricing scheme, and preprocessing algorithms (for the clique and star cases) that uses the dual information to speed up the generation of new columns. (4) We identify a set of additional valid inequalities that can be used to strengthen the proposed formulations. (5) We perform an extensive computational set of experiments to validate the applicability of the proposed approach.

2 Problem Definition

Let $\mathcal{T}$ be the collection of node structures in G from which the set of critical node structures is to be selected. The kind of elements in $\mathcal{T}$ may vary depending the context. For example, at its most basic form, if $\mathcal{T}$ is defined as the set of nodes, we have an instance of the critical node detection problem. Alternatively, if $\mathcal{T}$ is composed of more complex node structures such as paths, cliques, or stars, we have instances of the critical path, the critical clique, the critical star detection problems, respectively. In general, $\mathcal{T}$ can contain any user-defined subsets of nodes.

Notice that depending on the instance, the size of the elements in $\mathcal{T}$ might be different too (i.e., not all the cliques in a graph have the same size). Let $c(T)$ be a cost associated with the deletion of node structure $T \in \mathcal{T}$ and let b be a deletion budget. For the critical node case, we have that $\mathcal{T} = V$; thus, each $T \in \mathcal{T}$ is given by a unique node $i \in V$ and cost $c(T) = c_i$ is simply the deletion cost of node i. For more complex cases, such as for detecting critical cliques, $\mathcal{T}$ contains the set of all cliques in G and $c(T)$ is a function of the deletion costs of the nodes in T—e.g.,

$c(T) = \sum_{i \in V(T)} c_i$. Furthermore, in some contexts, the deletion cost $c(T)$ is often assumed to be one for all the elements. This latter version is frequently referred to as the *cardinality* version, as b represents an upper bound on the number of elements that can be deleted.

In general, most critical element detection problems aim to identify a subset of elements $\mathcal{K} \subseteq \mathcal{T}$—i.e., the critical elements—whose removal maximizes the degradation of a structural property of G, measured by a function $f\colon G \to \mathbb{R}$, while satisfying a budgetary constraint given by b. Among the different connectivity measures described in Section 1, we consider the following two: the size of the largest component and the total number of connected node pairs—which we attempt to minimize. A description of the objectives resulting from these measures follows:

Minimize the total number of connected node pairs (CNP): Given a graph $G = (V, E)$, a set of node structures $\mathcal{T}$ from G, and an integer b, the CNP objective looks for a set of node structures $\mathcal{K} \subseteq \mathcal{T}$ with a deletion cost of at most b such that, after removing the nodes and edges of all the node structures in $\mathcal{K}$, the total number of connected node pairs in the remaining graph is minimized:

$$\min \left\{ \sum_{H \in \Theta_{G[V \setminus V(\mathcal{K})]}} \binom{|V(H)|}{2} : \sum_{T \in \mathcal{K}} c(T) \leq b,\, \mathcal{K} \subseteq \mathcal{T} \right\}, \tag{1}$$

where $V(\mathcal{K}) = \bigcup_{T \in \mathcal{K}} V(T)$.

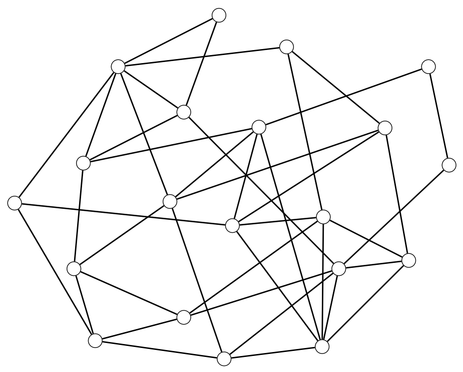
Minimize the size of the largest component (LC): Given a network $G = (V, E)$, a set of node structures $\mathcal{T}$ from G, and an integer b, the LC objective looks for a set of node structures $\mathcal{K} \subset \mathcal{T}$ with a deletion cost of at most b such that, after removing the nodes and edges of all the node structures in $\mathcal{K}$, the size of the largest component in the resulting graph is minimized:

$$\min \left\{ \max_{H \in \Theta_{G[V \setminus V(\mathcal{K})]}} |V(H)| : \sum_{T \in \mathcal{K}} c(T) \leq b,\, \mathcal{K} \subseteq \mathcal{T} \right\}. \tag{2}$$

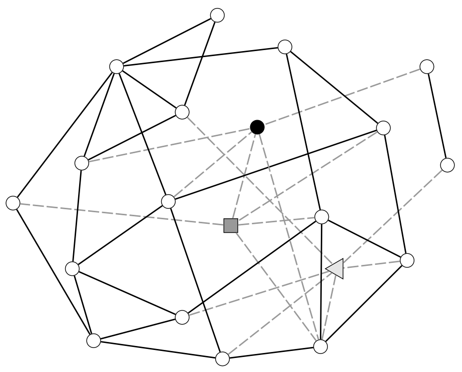
For illustrative purposes, consider the example depicted in Figure 1, in which we solve the problem of minimizing the total number of connected node pairs by removing three types of critical node structures: nodes (Figure 1(b)), cliques (Figure 1(c)), and stars (Figure 1(d)). Figure 1(a) shows the original graph G composed of $n = 20$ nodes and $m = 40$ edges and Figures 1(b), 1(c), and 1(d) depict the corresponding optimal solutions and resulting graphs after the removal of three critical nodes, critical cliques and critical stars, respectively. These solutions result in a reduction in the total number of connected node pairs from $\binom{20}{2} = 190$ to 106 for the node case (i.e., $\binom{15}{2} = 105$ connected node pairs in the largest component and $\binom{2}{2} = 1$ in the smallest), 28 for the clique case (i.e., $\binom{7}{2} = 21$ connected node pairs in the largest component, $\binom{4}{2} = 6$ in the second to largest component, and $\binom{2}{2} = 1$ in the smallest) and 0 for the star case (i.e., the remaining graph is composed of isolated nodes). In Figure 1(c), two of the cliques removed are of size two (the black circles and light gray triangles) and one of size three (the gray squares), whereas in Figure 1(d), the sizes of the stars deleted are four (gray squares), three (light gray triangles), and five (black circles).

Notice that, given the results of small examples like those of Figure 1, it is possible to prematurely draw the conclusion that removing node structures that are either large or that have many neighbors will always yield the largest possible disconnection of G. Clearly, removing a node structure T' with many nodes (assuming G and T' are connected) reduces the number of connected node pairs by at least $\binom{|V(T')|}{2} + |V(T')||V \setminus V(T')|$, where the first term corresponds to the connections within T' and the second term accounts for the connections between the nodes in T' and the nodes outside of T' that would be severed if T' is removed.

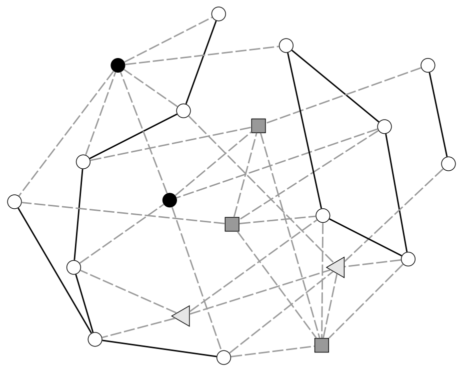
Figure 1: Examples of the effect on the number of connected node pairs when a set of three different critical node structures are removed from a graph. The node structures are depicted with black circles, gray squares, and light gray triangles, and the remaining nodes are colored white. The corresponding edges that are removed with the critical structures are shown dashed.



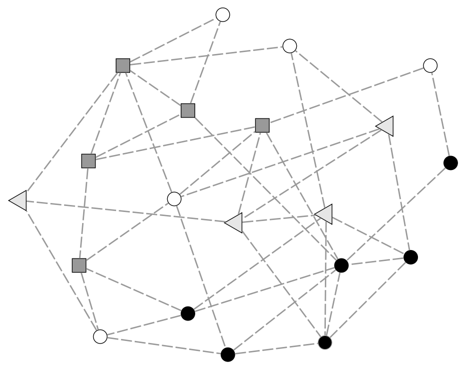
(a) $G = (V, E)$ with $n = 20$, $m = 40$, and 190 connected node pairs.



(b) Three critical nodes whose removal results in 106 connected node pairs on the remaining graph.

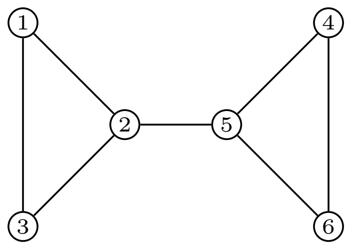


(c) Three critical cliques whose removal results in 28 connected node pairs on the remaining graph.

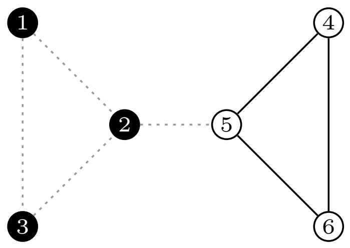


(d) Three critical stars whose removal results in 0 connected node pairs on the remaining graph (all remaining nodes become isolated).

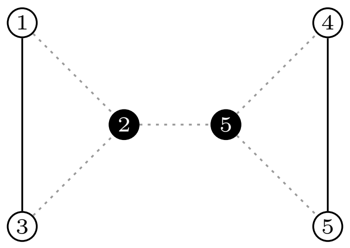
Figure 2: Examples of suboptimal solutions produced by greedily removing critical cliques.



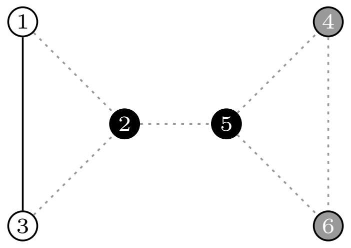
(a) $G = (V, E)$ with $n = 6$, $m = 7$, and 15 connected node pairs.



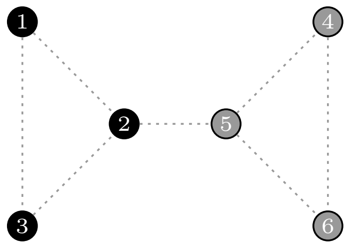
(b) Heuristic solution with three connected node pairs obtained after removing a largest clique.



(c) True optimal solution having two connected node pairs.



(d) Heuristic solution with one connected node pair after sequentially removing the two cliques with the largest number of neighbors.



(e) True optimal solution having no connected node pairs.

Similarly, removing a node structure T'' with many neighbors could potentially sever many paths passing through T'' that connect other node pairs in G. Thus, to maximally disconnect the graph, we can intuitively find a solution to these problems by greedily removing either the largest node structure (i.e., $T' = \arg\max_{T \in \mathcal{T}}\{|V(T)|\}$) or the node structure with the largest number of neighbors (i.e., $T'' = \arg\max_{T \in \mathcal{T}}\{|\bigcup_{i \in V(T)} \mathcal{N}_G(i) \setminus V(T)|\}$) and then reiterate over the residual graph until budget b is exhausted. Nevertheless, as shown in Figure 2, this approach often yields suboptimal solutions. Given the graph depicted in Figure 2(a), suppose we want to remove one critical clique to minimize the number of connected node pairs. Assuming that $b = 1$ and $c(T) = 1$ for each clique $T \in \mathcal{T}$, notice that removing a largest clique (colored in black in Figure 2(b)) produces a solution that is dominated by one that removes a smaller clique (see Figure 2(c)).

Furthermore, suppose that we want to remove two cliques to minimize the number of connected node pairs. For this example, $b = 2$ and $c(T) = 1$, for each clique $T \in \mathcal{T}$. Note that the black clique of size two has the largest number of neighbors in G (see Figure 2(d)). Then, following the greedy approach, we remove this clique and any of the residual cliques of size two like the one in gray in Figure 2(d). This yields a solution with one connected node pair, whereas the true optimal solution presented in Figure 2(e) consists of removing both cliques of size three. Similar examples for other critical node structures as well as for the case in which the size of the largest component of the residual graph is minimized can be easily generated. These greedy algorithms not only fail to produce optimal solutions; it is easy to see that one can generate instances for which the difference between the optimal and heuristic solutions is arbitrarily large.

3 Computational Complexity

The problem of removing a set of nodes so that the resulting graph satisfies a given structural property has been extensively studied in recent years. One of the most relevant results on this topic is given by [43]. In this work the authors proved that a node deletion problem is NP-hard whenever the structural property is hereditary and non-trivial. From this result, several authors have proven that many variations of critical node detection problem, including those that have CNP and LC as objectives functions, are NP-hard (e.g., [64, 73]). Other complexity results regarding critical node problems can be found in [1, 6, 23, 63, 64].

Contrary to the case of the critical nodes, the complexity of removing larger node structures has received significantly less attention. To the best of our knowledge, the only complexity result of this kind is given by [30]. In that work, the authors proved that the problem of removing a critical path so that the size of the largest component is minimized is also NP-hard. In the subsequent part of this section, we provide some complexity results for removing other types of critical node structures. We will focus our attention on the cardinality version of these problems, as these results can be easily extended for the weighted counterparts. Most of the following proofs are derived from the fact that many critical element detection problems generalize some covering and partitioning problems. For instance, as noted in [1], the vertex (node) cover problem is a particular case of the critical node detection problem because any set of b nodes whose removal completely disconnects the graph induces a node cover too.

We follow the standard approach used by [29] and define first a decision (or recognition) version of these problems. Clearly, because of the following remark, such decision problems are in NP.

Remark 1 *Computing the size of the largest component and the total number of connected node pairs of a graph can be done in linear time by means of a depth first search algorithm [36].*

3.1 Complexity of Detecting Critical Cliques

First, we develop some results for the case in which the set of node structures $\mathcal{T}$ comprises the set of all possible cliques in G. We will prove that this problem is NP-hard for both CNP and LC objectives. The formal definition of the decision version follows.

Problem 1 ***Critical Clique Detection Problem (CCP)***

INSTANCE: A simple non-empty graph $G = (V, E)$ and two nonnegative integers r and b.
QUESTION: Is there a set $\mathcal{K}$ of at most b cliques so that a connectivity measure over $G[V \setminus V(\mathcal{K})]$ is no greater than r.

Theorem 1 *CCP for both the CNP and LC measures is strongly NP-complete.*

The proof of this theorem, which can be found in the Appendix Part A, applies a reduction from k-COLORABILITY to CCP.

3.2 Complexity of Detecting Critical Stars

We now provide complexity results for the case in which the set of node structures $\mathcal{T}$ comprises the set of all possible stars in G. We will also prove that this problem is NP-hard for the objective functions described in Section 2.

Problem 2 ***Critical Star Detection Problem (CSP)***

INSTANCE: A simple non-empty graph $G = (V, E)$ and two nonnegative integers r and b.
QUESTION: Is there a set $\mathcal{K}$ of at most b stars so that a connectivity measure over $G[V \setminus V(\mathcal{K})]$ is no greater than r.

Theorem 2 *CSP for both the CNP and LC measures is strongly NP-complete*

The proof of this theorem, which can also be found in the Appendix Part A, uses a reduction from Dominating Set to CSP.

3.3 Complexity of Detecting Critical Connected Subgraphs

We now prove a more general complexity result, in which the elements of set $\mathcal{T}$ induce connected subgraphs of G. We show that detecting critical disjoint connected subgraphs is NP-hard as well.

Problem 3 ***Critical Connected Subgraph Detection Problem (CCSP)***

INSTANCE: A simple non-empty graph $G = (V, E)$, a set of connected disjoint subgraphs $\mathcal{T}$, and two nonnegative integers r and b.
QUESTION: Is there a set $\mathcal{K} \subset \mathcal{T}$ of at most b connected subgraphs so that a connectivity measure over $G[V \setminus V(\mathcal{K})]$ is no greater than r.

Theorem 3 *CCSP for both the LC and CNP measures is strongly NP-complete.*

The proof of this theorem, which can be found in the Appendix Part A, presents a reduction from Exact Cover to CCSP.

3.4 Inapproximability Results

Theorems 1, 2, and 3 show that, unless P=NP, it is hard to recognize if the optimal solution for a given instance of CCP, CSP, and CCSP, is zero for the CNP connectivity measure. For the case of the LC measure, it is easy to adapt proofs of Theorems 1 and 2 to show that k-colorability and dominating set can be transformed into instances of CCP and CSP, respectively, with $r = 0$. The implication of these results is that approximating CCP, CSP, and CCSP within any factor is NP-hard too. For the critical node problem, this result was previously mentioned in [1] .

4 Mixed-Integer Formulations

We first introduce general formulations defined for a generic set of node structures $\mathcal{T}$ and present an overview of the solution techniques proposed to solve the resulting problems. Subsequently, in Section 7, we provide thorough details for two common node structures (i.e., cliques and stars).

Given a network $G = (V, E)$, a set of node structures $\mathcal{T}$ from G, and an integer b, let a_i^T be a parameter that indicates with one if $i \in V$ is in $T \in \mathcal{T}$, and with zero if otherwise. Let $z_T \in \{0, 1\}$ be a variable that takes the value of one if node structure $T \in \mathcal{T}$ is removed from G, and zero, otherwise. In other words, $z_T = 1$, if and only if, $T \in \mathcal{K}$. Additionally, let $x_i \in \{0, 1\}$ be a variable that takes the value of one if node i is removed as a part of a critical node structure, and zero otherwise. We also define a variable y_{ij} for every pair $i, j \in V$ so that $y_{ij} = 1$, if and only if, i and j remain connected in $G[V \setminus V(\mathcal{K})]$. In other words, $y_{ij} = 1$ if neither node i nor j belong to any node structure $T \in \mathcal{K}$ and at least one path connecting i with j remains in $G[V \setminus V(\mathcal{K})]$. Note that by definition y_{ij} and y_{ji} represent the same variable for all $\{i, j\} \in \binom{V}{2}$; henceforth, we will use both versions interchangeably as needed without any further clarification.

4.1 Identifying Connected Node Pairs

One of the crucial components of our formulation is the set of constraints that are imposed to quantify the connectivity of the residual graph; more specifically, constraints that account for the number of connected node pairs. Identifying such connected node pairs within a subgraph is a task that not only arises when detecting critical elements; it is also common in graph partitioning [31], clustering [34], and network analysis [15, 71]. In the critical element detection context, such a set of constraints must enforce the relationship that exists between variables $\mathbf{x}$ and $\mathbf{y}$. In other words, for any given subset of nodes N that is removed from a graph G, $\mathbf{x} \in \{0,1\}^{|V|}$ represents characteristic vector of N (i.e., $x_i = 1$ if $i \in N$ and $x_i = 0$ if $i \in V \setminus N$) and $\mathbf{y} \in \{0,1\}^{|\binom{V}{2}|}$ is the indicator vector of the connected node pairs in $G[V \setminus N]$ (i.e., $y_{ij} = 1$ if nodes i and j remain connected in $G[V \setminus N]$ and $y_{ij} = 0$, otherwise). Among all the possible ways of modeling the relationship between variables $(\mathbf{x}, \mathbf{y})$ using linear constraints, we present three that are commonly used in the literature.

An important remark should be made about the following sets of constraints. Since the objective function of the problems at hand minimizes the connectivity of the residual graph, it is a common practice to add only the constraints that enforce y_{ij} to be one when the corresponding nodes i and j are connected; thus omitting the constraints that restrict y_{ij} to be zero in case no path exists between i and j (see, [6, 72]). The reason for this omission is that for the CNP connectivity measure the objective function will enforce $y_{ij} = 0$ in case i and j are disconnected; and for the LC measure, any optimal solution that does not satisfy the latter condition can be easily transformed into one that does. The constraint sets presented below thus describe solution sets that may contain values of $(\mathbf{x}, \mathbf{y})$ in which $y_{ij} = 1$ for some pair of nodes i and j that are disconnected in $G[V \setminus N]$. Nevertheless, none of those solutions underestimates the CNP or LC measures (in fact the opposite occurs). In other words, none of those solutions will ever be considered to be optimal for both the CNP and LC measures.

Let Ω be the set of all feasible values that pair $(\mathbf{x}, \mathbf{y})$ can take for describing the connectivity of any residual subgraph. We will refer to Ω as the *subgraph connectivity polytope*. Any of the three following constraint sets can be used to guarantee that the $(\mathbf{x}, \mathbf{y})$ values corresponding to all optimal solutions for the problem of detecting critical node structures of a graph belong to Ω.

- *Transitive Constraint Set* (also known as triangular constraints): The following set of constraints, initially popularized by [6], are adapted from the clique partitioning context [31]. This formulation enforces the transitive relationship that exists between any three nodes that are connected. That is, if i is connected to j and j is connected to k, i and k must also be connected. A similar version of these constraints also appears in [49].

$$y_{ij} + x_i + x_j \geq 1, \ \{i,j\} \in E; \tag{3}$$

$$y_{ij} + y_{jk} - y_{ik} \leq 1, \ \{i,j,k\} \in \binom{V}{3}; \tag{4}$$

$$x_i \in \{0,1\}, \ i \in V; \quad y_{ij} \in \{0,1\}, \ \{i,j\} \in \binom{V}{2}. \tag{5}$$

  Here, constraints (3) ensure that if two adjacent nodes i and j are not removed from the graph, they must be connected. Also, constraints (4) enforce the transitive relationships described above. The total number of constraints in this formulation is $O(n^3)$.

- *Path Constraint Set* (see, [22]): Given a pair of nodes $\{i,j\} \in \binom{V}{2}$, let $\mathcal{P}^{ij}$ be the set of all simple paths connecting i with j in G. Clearly, unless at least one node from each path

$P \in \mathcal{P}^{ij}$ is removed from G, nodes i and j will remain connected. Since the set of paths connecting any two nodes in a graph could grow exponentially, to use this formulation, the path inequalities (6) must be separated iteratively. We provide further information about how to handle these constraints in Section 5.1. Essentially, the current value of x_k can be viewed as the cost of traversing node $k \in V$. Then, if there is a path between any two nodes i and j whose cost plus the current value of y_{ij} is less than one, the inequality corresponding to such a path is violated and must be added to the formulation.

$$\sum_{k \in V(P)} x_k + y_{ij} \geq 1, \ \{i,j\} \in \binom{V}{2}, P \in \mathcal{P}^{ij}; \tag{6}$$

$$x_i \in \{0,1\}, \ i \in V; \quad y_{ij} \in \{0,1\}, \ \{i,j\} \in \binom{V}{2}. \tag{7}$$

For an edge variation of the path inequalities, see [51].

- *Neighborhood Constraint Set* (see, [72]): Given a pair of non-adjacent nodes $\{i,j\} \in \binom{V}{2}$, if i is connected to j, then j must also be connected to at least one neighbor of i. Therefore, if i and j are not removed (i.e., $x_i = x_j = 0$) and both remain connected, then $0 < \sum_{k \in \mathcal{N}_G(i)} y_{jk} \leq |\mathcal{N}_G(i)|$ and $0 < \sum_{k \in \mathcal{N}_G(j)} y_{ik} \leq |\mathcal{N}_G(j)|$. One of the advantages of this formulation is that there are in total $O(n^2)$ constraints.

$$y_{ij} + x_i + x_j \geq 1, \ \{i,j\} \in E; \tag{8}$$

$$y_{ij} + x_i \geq \frac{1}{|\mathcal{N}_G(i)|} \sum_{k \in \mathcal{N}_G(i) \setminus \{j\}} y_{jk}, \ \{i,j\} \in \binom{V}{2}; \tag{9}$$

$$x_i \in \{0,1\}, \ i \in V; \quad y_{ij} \in \{0,1\}, \ \{i,j\} \in \binom{V}{2}. \tag{10}$$

Other alternatives to quantify connectivity describe Ω using bilevel formulation typically adapted from other types of network interdiction problems (e.g., shortest path interdiction problems [79]). Two of these types of formulations can be found in [22, 64]. We restrict our discussion to the three we fully described above as those are the ones that have permeated the literature the most [21, 23, 30, 33, 52, 55, 73]. Moreover, bilevel optimization based models cannot be easily adapted to our settings and they do not demonstrate any significant advantages over the models described above.

4.2 General Formulations

The problem of detecting critical node structures of a graph for the CNP connectivity measure admits the following formulation:

$$\min \sum_{\{i,j\} \in \binom{V}{2}} y_{ij} \tag{11}$$

$$\text{s.t.} \ \sum_{T \in \mathcal{T}} c(T) z_T \leq b; \tag{12}$$

$$\sum_{T \in \mathcal{T}} a_i^T z_T = x_i, \ i \in V; \tag{13}$$

$$(\mathbf{x}, \mathbf{y}) \in \Omega; \tag{14}$$

$$\mathbf{z} \in \{0,1\}^{|\mathcal{T}|}, \tag{15}$$

where, the objective (11) minimizes the total number of connected node pairs. Constraint (12) guarantees that the cost of removing node structures does not exceed budget b. Constraints (13) force $x_i = 1$ if one of the node structures that contain i is removed. Finally, constraint (14)—which could be enforced by any of the previously described set of constraints—guarantees that $\mathbf{x}$ and $\mathbf{y}$ belong to the connectivity set Ω. Similarly, a valid formulation for the LC connectivity measure follows:

$$\min\ u \tag{16}$$
$$\text{s.t.}\quad (12)\text{-}(15); \tag{17}$$
$$\sum_{j\in V} y_{ij} + (1 - x_i) \leq u, \quad i \in V, \tag{18}$$

where the objective (16) minimizes the size of the largest component. Notice that two nodes are connected if they belong to the same component. Therefore, the size of the component that contains node i that has not been deleted is equal to the total number of nodes that i is connected to plus one—the node itself.

Notice that, since $x_i \leq 1$ for all $i \in V$, constraints (13) also force all deleted critical node structures to be disjoint, which is not always a requirement for the critical node structure detection problem. In case the critical node structures are allowed to share nodes, we have the following observations. First, if the set of node structures $\mathcal{T}$ is closed under inclusion (i.e., if $T_1 \subseteq T_2$ and $T_2 \in \mathcal{T}$, then $T_1 \in \mathcal{T}$), constraints (13) will suffice because any solution $\mathcal{K}$ with overlapping critical node structures can be transformed into a solution $\mathcal{K}'$ with disjoint critical node structures (see the proofs of Theorems 1 and 2 in the Appendix Part A for the clique and star cases). On the other hand, if $\mathcal{T}$ is not closed under inclusion, constraints (13) can be replaced by the following constraints.

$$\sum_{T\in\mathcal{T}} a_i^T z_T \leq x_i |V(T)|, \ i \in V. \tag{19}$$

Among the three connectivity constraint sets that were described above, we will use the path constraints in spite of resulting in a formulation that could grow exponentially large. As shown in Section 5.1, the separation algorithm can be solved in polynomial time during the execution of the solution procedure. Furthermore, it is easy to prove that the linear relaxation of the formulation that includes the path constraint set dominates the ones obtained by replacing those with either the transitive constraint set or the neighborhood constraint set. We will provide such a proof for the CNP connectivity measure. To this end, we will show that the optimal solution obtained by solving the linear relaxation for the path constraints is always feasible in the corresponding formulation when the transitive and neighborhood constraints are used. Further, we provide a specific instance for which the formulation with the path constraints produces a strictly better optimal linear relaxation than the other two variations.

Proposition 1 *Let $(\bar{\mathbf{z}}, \bar{\mathbf{x}}, \bar{\mathbf{y}})$ be an optimal solution of the linear relaxation of (11)-(15), in which (14) is defined by the path constraint set (6)-(7). Then $(\bar{\mathbf{x}}, \bar{\mathbf{y}})$ satisfies the linear relaxation of the formulation with the transitive constraint set (3)-(5).*

Proposition 2 *Let $(\bar{\mathbf{z}}, \bar{\mathbf{x}}, \bar{\mathbf{y}})$ be an optimal solution of the linear relaxation of (11)-(15), in which (14) is defined by the path constraint set (6)-(7). Then $(\bar{\mathbf{x}}, \bar{\mathbf{y}})$ satisfies the linear relaxation of the formulation with the neighborhood constraint set (8)-(10).*

Propositions 1 and 2—whose proofs can be found in the Appendix Part A—showed that, in terms of the linear relaxation quality, the formulation with the path constraints is not worse than the ones with the transitive and the neighborhood constraints. The following example provides evidence of an instance in which the formulation with the path constraints produces a linear relaxation that is strictly better than the other two.

Example 1 *Consider an instance of the problem of detecting critical node structures in a graph* $G = (V, E)$*, where* $V = \{1, 2, 3, 4, 5\}$ *and* $E = \{\{1, 2\}, \{1, 4\}, \{1, 5\}, \{2, 3\}, \{3, 4\}\}$*. Let* $\mathcal{T} = V$*,* $b = 1$*, and* $c(T) = 1$*, for all* $T \in \mathcal{T}$*. For the CNP measure, the optimal solution of the linear relaxation of (11)-(15), using the path constraint set (6)-(7), is* $x_1 = y_{23} = y_{24} = y_{34} = 1$ *(the other variables are equal to zero), which is integral and has an objective value of 3. On the other hand, the optimal solutions of the linear relaxation of such a formulation when using the transitive constraint set (3)-(5) or the neighborhood constraint set (8)-(9) are* $x_1 = x_3 = y_{12} = y_{14} = y_{15} = y_{23} = y_{34} = 0.5$ *(the other variables are equal to zero); and* $x_1 = y_{23} = y_{34} = 1, y_{24} = 0.5$ *(the other variables are equal to zero); respectively, both having an objective value of 2.5.*

4.3 Disjoint Node Structures

One basic case that is important to consider separately occurs when set $\mathcal{T}$ comprises disjoint node structures. That is, for all pairs of node structures $T, T' \in \mathcal{T}$ so that $T \neq T'$, we have that $V(T) \cap V(T') = \emptyset$. Clearly, for this case $\bigcup_{T \in \mathcal{T}} V(T) \subseteq V$, implying that $|\mathcal{T}| \leq |V|$. We give a special consideration to sets of disjoint structures because the corresponding problem admits a natural transformation into a generalized version of the critical node problem via merging each node structure $T \in \mathcal{T}$ into one single node.

The proposed transformation (which follows a similar idea of the clique collapsing heuristic in [75]) constructs a graph $\hat{G} = (\hat{V}, \hat{E})$ as follows. For the given set of nodes V in G, let $\tilde{V}$ be the (possibly empty) set of nodes that do not belong to any of the node structures in $\mathcal{T}$. That is, the largest $\tilde{V} \subset V$ for which $\tilde{V} \cap V(T) = \emptyset$, for all $T \in \mathcal{T}$. Let $V^{\mathcal{T}}$ be a set of $|\mathcal{T}|$ nodes, each representing one of the node structure $T \in \mathcal{T}$, and let the node set of $\hat{G}$ be $\hat{V} = \tilde{V} \cup V^{\mathcal{T}}$.

We begin constructing set $\hat{E}$ by adding all the edges in E that connect pairs of nodes in $\tilde{V}$ in G (i.e., the edges in $E \cap \binom{\tilde{V}}{2}$). Then, we add an edge $\{i, j\}$ between node $i \in \hat{V}$ and the node $j \in V^{\mathcal{T}}$ that represents node structure T, if there exists at least one edge connecting i with a node of $V(T)$ in G. Furthermore, we add an edge $\{i, j\}$ between the nodes $i \in V^{\mathcal{T}}$ representing node structure T and $j \in V^{\mathcal{T}}$ representing node structure T', if there exists at least one edge connecting nodes in $V(T)$ and $V(T')$ in G.

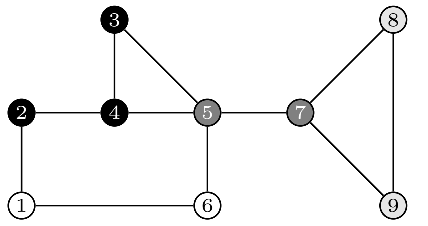
Figure 3 provides an example of the proposed transformation over an instance with a 9-node graph G and a node structure set $\mathcal{T} = \{\{2, 3, 4\}, \{5, 7\}, \{8, 9\}\}$ into $\hat{G}$. In this example, the transformed graph $\hat{G}$ has 5 vertices after merging the node structures in $\mathcal{T}$.

We now proceed to describe the formulation that results from the proposed transformation. We limit our discussion to the CNP connectivity measure, as a trivial extension exists for the LC connectivity measure as well. For this purpose, we will also use both variables $\mathbf{x}$ and $\mathbf{y}$ defined over $\hat{V}$ and $\binom{\hat{V}}{2}$, respectively.

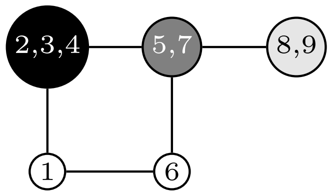
The cost of removing the nodes in $V^{\mathcal{T}}$ translates directly from the cost of removing the node structures. That is, the cost of removing node $i \in V^{\mathcal{T}}$ is equal to the cost of removing the corresponding node structure T node i represents. Furthermore, the nodes in $\tilde{V}$ are not assigned a removal cost as they do not belong to any node structure.

To calculate the number of node pairs that are disconnected whenever a node $i \in V^{\mathcal{T}}$ is removed, the objective function must account for: (1) the connected node pairs removed from within the

Figure 3: Graph transformation for an instance with disjoint node structures.



(a) Instance with a node structure set $\mathcal{T} = \{\{2,3,4\},\{5,7\},\{8,9\}\}$ depicted in black, gray, and light gray, respectivelly.



(b) Transformed graph G' by aggregating the node structures.

corresponding node structure T represented by i, and (2) the node pairs between the nodes in T and the nodes outside of T in G. For this purpose, for node $i \in V^{\mathcal{T}}$ representing node structure $T \in \mathcal{T}$, we define $d_i = \binom{|V(T)|}{2}$ to be the number of connected node pairs within i. Similarly, for node $i \in V^{\mathcal{T}}$ representing node structure $T \in \mathcal{T}$ and each node $j \in \hat{V}$ so that $i \neq j$, we define $s_{ij} = |V(T)|$, whenever $i \in \tilde{V}$ (this accounts for the connections between j and all the nodes in i) and $s_{ij} = |V(T)||V(T')|$ for the case in which $j \in V^{\mathcal{T}}$ and j represents node structure $T' \in \mathcal{T}$ (these account for the connections between the nodes representing two node structures). Observe that here, we are assuming T is connected; otherwise, d_i and s_{ij} are calculated considering the node pairs with respect to all the connected components of T. Finally, for the pair of nodes $\{i,j\} \in \binom{\tilde{V}}{2}$, we let $s_{ij} = 1$. Then, the resulting formulation follows:

$$\min \sum_{i \in V^{\mathcal{T}}} d_i(1 - x_i) + \sum_{\{i,j\} \in \binom{V'}{2}} s_{ij} y_{ij} \tag{20}$$

$$\text{s.t.} \sum_{i \in V^{\mathcal{T}}} c_i x_i \leq b; \tag{21}$$

$$(\mathbf{x}, \mathbf{y}) \in \Omega; \tag{22}$$

Notice that this approach does not work for other general cases where the node structures in $\mathcal{T}$ are not disjoint, as there is not a clear way of merging the nodes in $V(T)$ for all $T \in \mathcal{T}$. Given the similarities with the CNP, we will not explore this particular case any further. Instead, we will concentrate on the cases where $\mathcal{T}$ may contain node structures that are not disjoint.

5 Solution Strategy

The solution approach for solving formulations (11)-(15) and (16)-(18) depends on the characteristics of set $\mathcal{T}$ and the type of constraints used to enforce connectivity. For the simple cases where set $\mathcal{T}$ is computationally listable in a manageable time (e.g., its size is bounded by a low degree polynomial and parameters a_i^T and $c(T)$ are easily computable for all $T \in \mathcal{T}$), all variables $\mathbf{z}$ are available in advance without the need of additional pricing procedures (cf. Section 5.2). A few examples of these variations include the cases where $\mathcal{T}$ comprises the set of nodes (clearly listable in $O(n)$ time), the set of all maximal stars (listable in $O(m)$ time by traversing the adjacency list of the nodes), the set of all triangles (listable in $O(nm)$ time by enumerating all node triplets), the set of stars with a fixed bound κ on the number of leaves (listable in $O(n^\kappa)$ time by enumerating all $\binom{n}{\kappa}$ node tuples, assuming κ is constant and small), or, in general, the set of any "easily-identifiable" node structures with a bounded size κ (listable in $O(poly(\kappa) \cdot n^\kappa)$ time, assuming that κ is constant and small, and the validity certificate of the desired structure is obtainable in polynomial time on κ). Therefore, if $\mathcal{T}$, a_i^T, and $c(T)$ can be listed and easily computed, the only remaining part of the solution approach is the enumeration of the path constraints, which we discuss in Section 5.1. For this particular case, we also provide further enhancements in Section 6.

On the other hand, when solving problems with large sets $\mathcal{T}$ (e.g., the cases where $\mathcal{T}$ is the set of all cliques in G, the set of all stars in G, or any node structures whose enumeration involves algorithms with either high-degree polynomial or exponential complexity), we resort to the use of a column generation approach to overcome the burden of fully enumerating all variables $\mathbf{z}$ and parameters $c(T)$ and a_i^T for all $T \in \mathcal{T}$. The procedure starts with a smaller subset of node structures and sequentially introduces more of them in case they are needed to declare optimality. Along with the algorithm for separating the path constraints and the pricing procedure described in Sections 5.1 and 5.2, respectively, we embed the solution approach within a branch-price-and-cut scheme [7, 45] that works as follows: at each node of the branch-and-bound tree, depending on the desired objective, we solve a linear relaxation of (11)-(15) or (16)-(18) defined over a manageable subset of node structures $\mathcal{T}' \subset \mathcal{T}$. We refer to these formulations as the *master problems* (MPs, and more specifically, MP_{CNP} and MP_{LC}, respectively). If the current set $\mathcal{T}'$ is not sufficient to declare optimality, we proceed to price additional node structures in $\mathcal{T} \setminus \mathcal{T}'$ via column generation. Then, if the optimal solution at the end of this phase is not integral, we branch to reduce the solution space, thus eliminating undesired fractional solutions. Further, whenever a candidate integer solution is found, we separate additional violated path constraints (if any exist). We stop when all the branches of the branch-and-bound tree are processed and no further path inequality is required.

5.1 Separating the Path Constraints

As mentioned in Section 4.1 the number of path constraints can grow prohibitively large even for sparse graphs; therefore, we use an iterative enumeration in the so-called "lazy" fashion—i.e., we begin with an empty set of path constraints that is dynamically populated by identifying (separating) path constraints that are violated by the integer solutions found in the branch-and-bound tree. Given a candidate integer solution of the problem $(\bar{\mathbf{x}}, \bar{\mathbf{y}}, \bar{\mathbf{z}})$, the path constraints can be separated via solving an all-pairs shortest paths problem (APSP) [3] with node costs rather than edge costs. For each APSP run, the cost of traversing node $i \in V$ is set to $\bar{x}_i$; then, if the cost of the shortest path between any two nodes i and j plus the current value of $\bar{y}_{ij}$ is less than one, the corresponding inequality of such a path is violated and must be added to the formulation. Clearly, if the constraint associated with the shortest path between i and j is not violated by the current solution, any other constraint associated with a longer path would not be violated either.

Proposition 3 *Separating the path constraints for integer values of* $(\bar{\mathbf{x}}, \bar{\mathbf{y}}, \bar{\mathbf{z}})$ *takes* $O(nm)$ *time.*

The above implementation can be adapted for possibly adding simultaneously several violated inequalities for different node pairs. Notice that a path inequality between node i and j is violated if $\bar{y}_{ij} = 0$ and there is a path of cost zero between such nodes. Thus, in the step of Dijkstra's algorithm, in which the distance labels are updated, whenever the distance to a node is set to zero, we add the constraint associated with the respective path if $\bar{y}_{ij} = 0$. Furthermore, in our experiments, we have noticed that introducing path constraints that are violated by fractional solutions can also speed up the running time of the algorithm and decrease substantially the number of nodes of the branch-and-bound tree. The essence of the separation process for fractional solutions is exactly the same. The main difference is that the cost of traversing the nodes in the shortest path problem may take fractional values in the interval $[0, 1]$. Thus, finding violated inequalities after solving the linear relaxations of the branch-and-bound tree may require a different APSP implementation (e.g., Floyd-Warshall in $O(n^3)$ time, or a sequential Dijkstra's implementation using Fibonacci heaps in $O(m^2 \log n)$ time).

5.2 Pricing Node Structures

When solving critical node structure detection problems with large sets $\mathcal{T}$, it is generally common that the initial set $\mathcal{T}'$ used to define the MPs does not include an optimal set of critical node structures. Moreover, due to degeneracy, it is also possible that even if such a set of optimal critical structures exists in $\mathcal{T}'$, declaring the optimality of the MPs requires the consideration of additional elements from $\mathcal{T}\backslash\mathcal{T}'$ (i.e., expanding $\mathcal{T}'$ with additional candidate node structures). This is a known issue that generally occurs with most problems that are solved via column generation [7].

In the column generation procedure, at each iteration we are required to find whether there exists a new node structure $T \in \mathcal{T} \setminus \mathcal{T}'$ that improves the current solution. In other words, we aim to find a variable z_T such that its reduced cost is negative, or, from the dual perspective, a primal variable for which its corresponding dual constraint is violated. Consider the dual problems of the MPs, defined as DP_{CNP}, DP_{LC}.

$$
\begin{aligned}
(DP_{CNP}) : \max \quad & -b\alpha + \sum_{\{i,j\}\in\binom{V}{2}} \sum_{P\in\mathcal{P}^{ij}} \gamma_{ij}^P && (23)\\
\text{s.t.} \quad & -c(T)\alpha + \sum_{i\in V} a_i^T \beta_i \le 0, \ T \in \mathcal{T}'; && (24)\\
& -\beta_i + \sum_{\{j,k\}\in\binom{V}{2}} \sum_{\{P\in\mathcal{P}^{jk} | i\in V(P)\}} \gamma_{jk}^P \le 0, \ i \in V; && (25)\\
& \sum_{P\in\mathcal{P}^{ij}} \gamma_{ij}^P \le 1, \ \{i,j\} \in \binom{V}{2}; && (26)\\
& \alpha \ge 0; \quad \gamma_{ij}^P \ge 0, \ \{i,j\} \in \binom{V}{2}, \ P \in \mathcal{P}^{ij} && (27)
\end{aligned}
$$

and

$$
\begin{aligned}
(DP_{LC}) : \max \quad & -b\alpha + \sum_{\{i,j\}\in\binom{V}{2}} \sum_{P\in\mathcal{P}^{ij}} \gamma_{ij}^P && (28)\\
\text{s.t.} \quad & (24); (27); && (29)\\
& -\beta_i + \sum_{\{j,k\}\in\binom{V}{2}} \sum_{\{P\in\mathcal{P}^{jk} | i\in V(P)\}} \gamma_{jk}^P - \sigma_i \le 0, \ i \in V; && (30)\\
& \sum_{P\in\mathcal{P}^{ij}} \gamma_{ij}^P - \sigma_i - \sigma_j \le 0, \ \{i,j\} \in \binom{V}{2}; && (31)\\
& \sum_{i\in V} \sigma_i \le 1; && (32)\\
& \sigma_i \ge 0, i \in V. && (33)
\end{aligned}
$$

For these dual formulations, α is the dual of the budget constraint (12), β_i for $i \in V$ are the dual variables of constraints (13), γ_{ij}^P for $\{i,j\} \in \binom{V}{2}$ and $P \in \mathcal{P}^{ij}$ are the dual variables of the path constraints (6), and σ_i for $i \in V$ are the dual variables of constraints (18) for the formulation that uses the LC measure as objective. Furthermore, constraints (24) are complementary to the $\mathbf{z}$ variables, constraints (25) and (30) are complementary to variables $\mathbf{x}$, constraints (26) and (31) are complementary to variables $\mathbf{y}$ for both formulations, respectively, and constraint (32) is complementary to variable u of the LC formulation variation. Notice that (24) are the only constraints specifically associated with the node structures in $\mathcal{T}'$. This implies that, if we find new candidate structure $T \in \mathcal{T} \setminus \mathcal{T}'$ for which its corresponding constraint (24) is violated by the current solution—i.e., a node structure $T \in \mathcal{T} \setminus \mathcal{T}'$ for which $\sum_{i \in V} a_i^T \beta_i > c(T)\alpha$—we can augment $\mathcal{T}'$ with such a T. In consequence, the optimality conditions of the pricing stage can be defined as follows.

$$\sum_{i \in V} a_i^T \beta_i - c(T)\alpha \leq 0, \quad \forall T \in \mathcal{T} \setminus \mathcal{T}', \tag{34}$$

and, in particular,

$$\sum_{i \in V} a_i^T \beta_i \leq \alpha, \quad \forall T \in \mathcal{T} \setminus \mathcal{T}' \tag{35}$$

for the cardinality version of these problems where $c(T) = 1$ for all $T \in \mathcal{T}$. To identify promising node structures, we then need to solve a pricing subproblem that needs to be tailored for the specific characteristics of the elements in $\mathcal{T}$. In general, the proposed pricing subproblem has the following specific form:

$$\max_{\mathbf{a} \in \mathrm{A}^{\mathcal{T}}} \sum_{i \in V} \beta_i a_i - c(\mathbf{a})\alpha \tag{36}$$

where $\mathbf{a} \in \{0,1\}^{|V|}$ is the indicator vector of the nodes present in the node structure being generated, $\mathrm{A}^{\mathcal{T}}$ is the set of all possible realizations of $\mathbf{a}$, given the elements in $\mathcal{T}$, and $c(\cdot)$ is a function that accounts for the cost of the node structure being created. The objective function (36) maximizes the violation of (34)—i.e., finds a node structure with the smallest reduced cost, and constraints $\mathbf{a} \in \mathrm{A}^{\mathcal{T}}$ are the generic constraints that force $\mathbf{a}$ to represent a member of $\mathcal{T}$. If the cost of the node structures is given by the sum of individual costs on the nodes, the objective (36) results in $\sum_{i \in V}(\beta_i - \alpha c_i)a_i$. Furthermore, for the cardinality version of the problem, it is possible to simply maximize $\sum_{i \in V} a_i \beta_i$ and then compare the respective solution versus the value of α.

5.3 Branching Rule and Stabilization Procedure

Two additional components that are fundamental for the computational efficiency of any branch-and-price procedure are the branching rule and the stabilization procedure. First, in the branch-and-price context, it is well known [7] that applying the standard 0/1 branching (i.e., fixing a fractional variable to either zero or one) is undesirable because it produces highly unbalanced branching trees, and also because it requires introducing additional requirements—e.g., finding the kth best solution or introducing further constraints in the subproblems—to ensure that the pricing subproblem does not generate variables that have been fixed to zero in previous branches. Therefore, a different branching scheme that copes with these issues is highly desirable.

Second, solving the linear relaxations of formulations (11)-(15) and (16)-(18) via column generation could be a rather slow process because of the tailing-off effect prevalent in most of these types of applications [45]. This slow convergence is mainly caused by the degeneracy of these formulations and the instability of the dual variables throughout the execution, particularly in early iterations when fewer dual constraints have been generated and thus little information is known about the dual problem.

Since the quality of the proposed method depends so heavily on these two components, and because of the fact that the pricing subproblems can be rather difficult to solve (see Section 7), we have developed both a generic branching rule and a stabilization method that can be used for detecting any type of critical node structures, which, due to space limitations, are thoroughly described in the Appendix Part B.

6 Further Enhancements

In this section we introduce a variation of the path constraints (6) described in Section 4.1 and use it to derive two families of valid inequalities that strengthen the proposed formulations. The variation presented below can directly replace (6) when solving instances for which the set of node structures $\mathcal{T}$ is computationally listable in a manageable time. However, their use within the branch-price-and-cut scheme introduced in Section 5 requires additional considerations as the dual variables of these additional constrains have a strong effect on the structure of the pricing subproblem (see Section 6.3).

6.1 Path Constraints Revisited

Given a path $P \in \mathcal{P}^{ij}$ connecting a pair of nodes i and j in V, let $\mathcal{T}(P) = \{T \in \mathcal{T} : V(T) \cap V(P) \neq \emptyset\}$ be the set of node structures in $\mathcal{T}$ that contain nodes along path P. The following constraint set can be used in formulations (11)-(15) and (16)-(18) as an alternative to the path constraints (6). We name these constraints, *$\mathcal{T}$-path constraints.*

$$\sum_{T \in \mathcal{T}(P)} z_T + y_{ij} \geq 1, \ \{i,j\} \in \binom{V}{2}, P \in \mathcal{P}^{ij}. \tag{37}$$

The idea behind this variant coincides with the one of the ordinary path constraints (6) in that, if no node structure $T \in \mathcal{T}$ having nodes along a path $P \in \mathcal{P}^{ij}$ is removed from G, such a path also exists in $G[V \setminus V(\mathcal{K})]$, implying that nodes i and j remain connected after the set of critical node structures $\mathcal{K}$ is removed from G. The relationship between variables $\mathbf{x}$ and $\mathbf{z}$ given by constraints (13) implies that any solution $(\mathbf{x}, \mathbf{y}, \mathbf{z})$ for the CNP connectivity measure (or $(\mathbf{x}, \mathbf{y}, \mathbf{z}, u)$ for the LC connectivity measure) satisfying (37), satisfies (6) as well. To see this, simply replace (13) in (6) for any given path $P \in \mathcal{P}^{ij}$ to obtain

$$\sum_{k \in V(P)} \sum_{T \in \mathcal{T}} a_k^T z_T + y_{ij} \geq 1, \{i,j\} \in \binom{V}{2},$$

which is clearly satisfied by any solution satisfying (37), as $T \in \mathcal{T}(P)$ implies $a_k^T = 1$ for at least one node $k \in P$. In other words, the $\mathcal{T}$-path constraints produce a linear relaxation that is at least as good as the one with the ordinary path constraints. Nevertheless, formulations (11)-(15) and (16)-(18) with (37) instead of (6) generally produce tighter relaxations. The following example provides evidence of an instance for the CNP measure in which the formulation with the $\mathcal{T}$-path constraints produces a linear relaxation that is strictly better than the one with the ordinary path constraints.

Example 2 *Consider an instance of the problem of detecting critical node structures in the graph $G = (V, E)$ depicted in Figure 2(a), where $V = \{1, 2, 3, 4, 5, 6\}$ and $E = \{\{1, 2\}, \{1, 3\}, \{2, 3\}, \{2, 5\}, \{4, 5\}, \{4, 6\}, \{5, 6\}\}$. Let $\mathcal{T} = \{\{1, 2, 3\}, \{2, 5\}, \{4, 5, 6\}\}$, $b = 1$, and $c(T) = 1$, for all $T \in \mathcal{T}$. For the CNP measure, the optimal solution of the linear relaxation of (11)-(15), using the path constraint set (6)-(7), the optimal solution is $z^*_{\{1,2,3\}} = z^*_{\{4,5,6\}} = 1/2$, $z^*_{\{2,5\}} = 0$, $x^*_i = 1/2$, for all $i \in V$, and $y^*_{ij} = 0$, for all $\{i,j\} \in \binom{V}{2}$, with an objective value of zero. On the other hand, the optimal solutions of the linear relaxation of such a formulation when using the $\mathcal{T}$-path constraint set (37) yields the integral optimal solution $z^*_{\{1,2,3\}} = z^*_{\{4,5,6\}} = 0$, $z^*_{\{2,5\}} = x^*_2 = x^*_5 = 1$, $x^*_i = 0$, for all $i \in V \setminus \{2, 5\}$, $y^*_{13} = y^*_{46} = 1$, and $y^*_{ij} = 0$ for all $\{i,j\} \in \binom{V}{2} \setminus \{\{1, 3\}, \{4, 6\}\}$, with an objective value of 2.*

6.2 Valid Inequalities

We now provide two families of valid inequalities derived from the $\mathcal{T}$-path constraints. The proofs of the following results can be found in the Appendix Part A.

Proposition 4 *For any clique K of size 3 in G (a triangle), where $V(K) = \{i, j, k\}$, the following inequality is valid. We refer to these inequalities as 3-clique inequalities.*

$$\sum_{\{T\in\mathcal{T}:|V(T)\cap V(K)|=1\}} z_T + 2\sum_{\{T\in\mathcal{T}:|V(T)\cap V(K)|>1\}} z_T + y_{ij} + y_{ik} + y_{jk} \geq 2. \tag{38}$$

The total number of 3-clique inequalities is given by the number of cliques of size three (triangles) in G, which can be found in $O(mn)$ time. Furthermore, generating each of these inequalities requires checking how many out of the three nodes of the corresponding clique are contained in each of the node structures in $\mathcal{T}$. This can can be done in $O(|\mathcal{T}|M)$ time, assuming that M is the size of the largest node structure in $\mathcal{T}$. If the size of $\mathcal{T}$ is small, all the 3-clique inequalities can be directly added into the formulation.

Proposition 5 *For any chordless cycle C of size 4 in G, where $V(C) = \{i, j, k, l\}$ and $E(C) = \{\{i, j\}, \{j, k\}, \{k, l\}, \{i, l\}\}$, the following inequality is valid. We refer to these inequalities as 4-cycle inequalities.*

$$\sum_{\{T\in\mathcal{T}:|V(T)\cap V(C)|=1\}} z_T + 2\sum_{\{T\in\mathcal{T}:|V(T)\cap V(C)|>1\}} z_T + y_{ik} + y_{jl} \geq 2. \tag{39}$$

The set of chordless cycles of size four in G can be found in $O(n^4)$ time and, as for the 3-clique case, generating each of the corresponding inequalities requires checking how many of the nodes of the cycle are contained in each of the node structures in $\mathcal{T}$, which can be done in $O(|\mathcal{T}|M)$ time as well. However, the number of chordless cycles varies drastically depending on the graph's density. A complete enumeration of these inequalities can be computationally burdensome and thus limited for small instances.

Analyzing the strength of the above inequalities requires additional insights regarding the dimension of formulations (11)-(15) and (16)-(18), which is in general a difficult task because the dimension of these polyhedra highly depends on the set of node structures $\mathcal{T}$. In fact, in some cases, like when $\mathcal{T}$ comprises the set of all cliques, even finding the size of $\mathcal{T}$ can be notably hard. A preliminary analysis using computational tools for testing properties of polyhedral sets—i.e., Porta [16]—over small instances suggests that the $\mathcal{T}$-path, the 3-clique and the 4-cycle inequalities often induce facets of the these polyhedra. However, further analysis is required to make a definite conclusion.

6.3 Implications for the Column Generation

For the cases where $\mathcal{T}$ is sequentially generated via column generation, using the $\mathcal{T}$-path constraints, the 3-clique, or the 4-cycle inequalities pose a strong effect on the pricing subproblem and can severely complicate the generation of new node structures. The main difficulty arises from the fact that the dual values of these inequalities must be incorporated into the objective of the subproblem in case the node structure being generated contains nodes in the paths, cliques and cycles of these inequalities. This challenge is particularly severe for the 3-clique and 4-cycle inequalities because the pricing problem requires knowing the exact number of nodes of the clique or cycle contained in the node structure that is being generated. If the subproblem is solved using mathematical programming, it is possible to incorporate extra variables and constraints to model these additional requirements; however, the resulting formulation may become difficult to solve. The decision of which version of the path constraints to use—either (6) or (37)—and whether the 3-clique and 4-cycle inequalities are added in this context raises then a trade-off between the quality of the linear relaxation and the difficulty of the pricing subproblem.

A viable alternative is to use the original path constraints and strengthen the formulation with 3-clique and 4-cycle inequalities, but without lifting the coefficient of the newly generated node structures in subsequent iterations. In other words, some 3-clique inequalities are generated beforehand including exclusively the variables of the node structures of the initial set $\mathcal{T}'$ and then, it is assumed that the variables of all the subsequently generated node structures have a coefficient of zero in those inequalities. This could however reduce the impact of the 3-clique inequalities, particularly if the size of the initial set $\mathcal{T}'$ is small.

7 Examples: Detecting Critical Cliques and Critical Stars

In this section we provide directions on how to use the proposed framework for detecting two common types of node structures: cliques and stars. First, we demonstrate how to use the branch-price-and-cut approach for solving the problem of detecting critical cliques, i.e., the case in which the set of node structures $\mathcal{T}$ comprises all cliques in G. We discuss the procedure used to tackle the resulting pricing subproblem and describe a preprocessing algorithm that uses the dual information to speed up the solution process. Second, we provide further details on how to adapt the framework for detecting critical stars as well. We describe the special case in which the set of node structures $\mathcal{T}$ comprises all stars in G having sizes up to a given value κ.

7.1 Detecting Critical Cliques

For this specific case, we initialize set $\mathcal{T}'$ with all the cliques of sizes up to three nodes—which can be listed in $O(mn)$ time—and then we generate additional cliques via column generation, as described in Section 5.2. Furthermore, we use the original version of the path constraints (6) in the master problem because it allows us to use a specialized algorithm [57] to solve the resulting pricing subproblem. We focus on the cardinality version of the problem (i.e., all the cliques have a cost of one independently of their size), but highlight the fact that minor variations in the objective function would suffice for tackling other cases as well. Based on formulation (36), the subproblem for pricing new candidate cliques is:

$$\max \sum_{i \in V} \beta_i a_i - \alpha \tag{40}$$

$$\text{s.t.} \quad a_i + a_j \leq 1, \ \{i, j\} \in \binom{V}{2} \setminus E; \tag{41}$$

$$a_i \in \{0, 1\}, \ i \in V, \tag{42}$$

where a_i becomes a binary variable that takes the value of one if node $i \in V$ is in the new candidate clique and zero, otherwise. Notice that since the value of α is known, the objective can be simply set to $\sum_{i \in V} \beta_i a_i$ and then, the optimal value can be compared with α. That is, if the resulting optimal value is larger than α, then the corresponding clique can be included in $\mathcal{T}'$. Notice that the subproblem given by formulation (40)-(42) is that of finding a clique of maximum weight in G, where the weights are the dual variables of constraints (13). Finding maximum cliques in general graphs is known to be NP-hard [29], which implies that pricing new cliques could be potentially challenging for large instances. Nevertheless, it is possible to reduce the complexity of the pricing subproblem by taking advantage of the fact that α provides a nonnegative lower bound on the weight that candidate cliques must have to be considered for inclusion in $\mathcal{T}'$. Using this bound to preprocess the graph by removing unnecessary nodes (i.e., nodes that cannot be contained in cliques with a weight larger than α), and applying specialized algorithms (e.g., [57]) over the resulting graph can be quite fruitful in practice. Before presenting how to take advantage of this lower bound, we first describe how to propagate the branching decisions over the graph to solve the pricing subproblem.

Given a pair of nodes i and j in V, the proposed branching rule (see the Appendix Part B.1) may produce a `DIFF` branch over i and j enforcing $a_i + a_j \leq 1$, or a `SAME` branch enforcing $a_i = a_j$. Instead of adding these requirements as constraints in formulation (40)-(42), we solve the pricing subproblem over a modified graph $\hat{G} = (\hat{V}, \hat{E})$ that ensures these branching constraints are enforced by construction. We initialize $\hat{G}$ as a copy of G and then alter its node and edge sets as follows. The `DIFF` requirements are enforced by removing edge $\{i, j\}$ from $\hat{E}$, which eliminates from the graph all cliques containing both nodes. On the other hand, the `SAME` requirements are enforced either by removing both nodes from $\hat{V}$, in case edge $\{i, j\} \notin \hat{E}$ (as non-adjacent nodes cannot be simultaneously in a clique), or by merging both nodes into a new node k, otherwise. If i and j are merged into a new node k, such a node inherits all the additional branching constraints of both i and j, the weight β_k is set to $\beta_i + \beta_j$, and its neighborhood is set as the intersection of the neighborhoods of i and j, updating $\hat{E}$, accordingly (i.e., $\mathcal{N}_{\hat{G}}(k) = \mathcal{N}_{\hat{G}}(i) \cap \mathcal{N}_{\hat{G}}(j)$).

After generating graph $\hat{G}$ from the branching constraints we can then use the dual bound given by α to remove nodes and edges from $\hat{G}$ that cannot belong to good candidate cliques. The preprosessing procedure, presented in Algorithm 1, can be described in three parts. The first part (see steps 1-5) removes any node $i \in \hat{V}$ with nonpositive weight β_i from $\hat{G}$, as any clique that contains i either increases or at least retains its weight if node i is removed. The second part (see steps 7-11) removes any node $i \in \hat{V}$ for which the sum of its weight and that of its neighbors is less than α. Since $\beta_i + \sum_{j \in \mathcal{N}_{\hat{G}}(i)} \beta_j$ is a bound on the weight of the cliques containing i, $\beta_i + \sum_{j \in \mathcal{N}_{\hat{G}}(i)} \beta_j \leq \alpha$ implies that node i does not belong to any clique worth extending $\mathcal{T}'$ with. The last part (see steps 12-16) tests if two adjacent nodes i and j can be simultaneously in a candidate clique. As with the second part, if the sum of β_i, β_j and the weights of the nodes adjacent to both i and j is less than α, no clique containing both nodes i and j is worth adding to $\mathcal{T}'$. Parts two and three are repeated until no further change in $\hat{G}$ can be made. One possible outcome of Algorithm 1 is an empty graph $\hat{G}$, which means that the current solution is optimal as no further good candidate clique exists.

Proposition 6 *The time complexity of Algorithm 1 is* $O(m^2 n + mn^2)$

Proof. The first part (steps 1-5) is done in $O(n)$ time. Part two (steps 7-11) requires traversing the adjacency lists of all the nodes, which takes $O(m)$ time. Part three (steps 12-16) requires finding the intersection of two adjacency lists per edge, which can be done in $O(n)$ time (assuming the adjacency lists are kept sorted), requiring thus $O(mn)$ time in total. Parts two and three are repeated at the most $O(n + m)$ times because each iteration of the loop removes either a node or an edge. The time complexity is then $O(m^2 n + mn^2)$.

In practice, Algorithm 1 is rather fast and makes a significant difference in the solution time of the overall branch-price-and-cut algorithm. It drastically reduces the computational time required to solve the pricing problem, specially during late stages after several cliques have been added to $\mathcal{T}'$, avoiding unnecessary calls to a rather complex subproblem. Finally, as mentioned before, we use a specialized algorithm [57] to solve the resulting subproblem.

Algorithm 1 A procedure that preprocesses the graph $\hat{G}$ used for solving the pricing subproblem. It removes nodes and edges that cannot belong to good candidate cliques.

Input: Graph $\hat{G} = (\hat{V}, \hat{E})$ and the current value of the dual variables α and β.
Output: An updated version of graph $\hat{G}$.

```
 1: for all $i \in \hat{V}$ do
 2:   if $\beta_i \leq 0$ then
 3:     $\hat{V} \leftarrow \hat{V} \setminus \{i\}$
 4:   end if
 5: end for
 6: repeat
 7:   for all $i \in \hat{V}$ do
 8:     if $\beta_i + \sum_{j \in \mathcal{N}_{\hat{G}}(i)} \beta_j \leq \alpha$ then
 9:       $\hat{V} \leftarrow \hat{V} \setminus \{i\}$
10:     end if
11:   end for
12:   for all $e = \{i, j\} \in \hat{E}$ do
13:     if $\beta_j + \beta_i + \sum_{k \in \mathcal{N}_{\hat{G}}(i) \cap \mathcal{N}_{\hat{G}}(j)} \beta_k \leq \alpha$ then
14:       $\hat{E} \leftarrow \hat{E} \setminus \{e\}$
15:     end if
16:   end for
17: until No further nodes or edges are removed from $\hat{G}$
18: return $\hat{G}$
```

7.2 Detecting Critical Stars

For this specific case, we assume that $\mathcal{T}$ comprises the set of all stars of sizes up to a given bound κ. If κ is small—say up to three leaves—we directly generate $\mathcal{T}$ in $O(n^{(\kappa+1)})$ time and then solve the resulting formulation as indicated in Section 5. Furthermore, since no column generation is required when κ is small, we use the $\mathcal{T}$-path constraints (37) and strengthen the formulation with the 3-clique (38) and the 4-cycle (39) inequalities. Alternatively, if κ is larger, we use the original version of the path constraints, as explained in Section 6.3, initializing $\mathcal{T}'$ with all the stars of sizes up to three nodes, and resorting to column generation for generating any subsequent larger star.

In this example, we will consider a case in which the cost of removing a star is a combination between the removing cost of its nodes minus a discount derived from a contagion process exerted from the hub of the star to its leaves. Let c_i be the cost of removing node $i \in V$ and $d_{ij} \leq c_i + c_j$ be the discount given by the contagion process from node i to node j. Then, the cost of removing a star $T \in \mathcal{T}$ centered at node i (i.e., $h(T) = i$) is $c(T) = c_i + \sum_{j \in l(T)} (c_j - d_{ij})$.

To price new star candidates we propose an iterative process that first fixes the hub of the star and then attempts to select the leaves from among its neighbors so as to produce a star with negative reduced cost. This process is repeated by fixing each of the n nodes in V as the star hub, thus potentially finding several new candidate stars per iteration. Based on the optimality conditions given by expression (34), a star $T \in \mathcal{T} \setminus \mathcal{T}'$ having node i as its hub is a good candidate to be included in $\mathcal{T}'$ if the following condition is satisfied:

$$\sum_{j \in V(T)} \beta_j - \alpha c(T) = \beta_i - \alpha c_i + \sum_{j \in l(T)} (\beta_j - \alpha(c_j - d_{ij})) > 0. \tag{43}$$

The pricing subproblem for finding a good candidate star centered at node i is then:

$$SP(i): \quad \beta_i - \alpha c_i + \max \sum_{j \in \mathcal{N}_G(i)} (\beta_j - \alpha(c_j - d_{ij}))a_j \tag{44}$$

$$\text{s.t.} \sum_{j \in \mathcal{N}_G(i)} a_j \leq \kappa; \tag{45}$$

$$a_j \in \{0,1\}, \; j \in \mathcal{N}_G(i), \tag{46}$$

where a_j is a binary variable that takes the values of one if neighbor $j \in \mathcal{N}_G(i)$ is selected to be a leaf of the star and zero, otherwise. Additionally, since the values of α and β_i are known, the optimal value of the optimization problem can be compared with $\alpha c_i - \beta_i$.

In the absence of branching constraints, formulation (44)-(46) can be solved by sorting the neighbors of i in descending order based on their objective's coefficients and then selecting up to κ nodes with largest (positive) objective coefficients. However, after branching constraints are introduced in the branch-and-bound tree, propagating such constraints into the subproblem may complicate the solution procedure. The propagation of the `SAME` branching constraints is relatively easy, as those can be handled by either merging or removing nodes in a similar fashion as the one explained for the clique case. The `DIFF` branching constraints are more difficult to handle, specifically the case in which the two nodes associated with the constraint are both potential leaves of the same star. Assuming the constraint's nodes are j and k, and that both of such nodes are in $\mathcal{N}_G(i)$, for a given hub node i, a constraint of the form $a_j + a_k \leq 1$ must be added in formulation (44)-(46) to ensure k and j are not simultaneously part of the same star. These additional constraints transform the pricing subproblem into a maximum weighted independent set problem over the graph induced by $\mathcal{N}_G(i)$, with a side constraint given by the size restriction κ. In practice, if the size of $\mathcal{N}_G(i)$ is small and the number of `DIFF` constraints is limited, the pricing subproblem can be efficiently solved. Finally, similar procedures to the ones described in steps 1-5 and 7-11 of Algorithm 1 can be applied to reduce the computational burden of finding candidate stars.

8 Computational Experiments

We divide the computational study into two parts. First, we evaluate the performance of the proposed framework for tackling the case in which the set of candidate node structures is generated iteratively from $\mathcal{T}$. To this end, we apply the branch-price-and-cut scheme to solve several instances of the cardinality version of the critical clique problem for both the CNP and LC objectives, as described in Section 7.1. We are particularly interested in analyzing the impact of the path constraint separation and the variable pricing algorithms over the total execution time.

Second, we solve instances in which the set of possible critical node structures $\mathcal{T}$ is small enough to be directly incorporated in the proposed formulation. For this part, we solve instances of the critical star problem, as explained in Section 7.2, restricting set $\mathcal{T}$ to comprise stars with zero, one, and two leaves. We set the cost of removing such stars to be 100, 175, and 250, respectively; i.e., we set $c_i = 100$, for all $i \in V$ and $d_{ij} = 25$, for all $\{i,j\} \in E$ (see Section 7.2 for the specific definition of c and d in the critical star detection context). The objective is to compare the strength of the path constraints versus that of the $\mathcal{T}$-path constraints, and to analyze the impact over the execution time of the 3-clique and 4-cycle inequalities. For these experiments, we only use the CNP objective.

8.1 Hardware, Software, and Test Instances

The computational study was performed on a computing cluster equipped with twelve nodes, each having a 12-core Intel Xeon E5-2620 v3 2.4GHz processor, 128 GB of RAM, and running Linux x86_64, CentOS 7.2. All the formulations and algorithms were implemented in C, using SCIP [11] to develop the proposed branch-price-and-cut framework. The instances were solved by SCIP, combined with CPLEX 12.6.2 as the linear optimizer. A time limit of 7,200 seconds per instance was set for the computations.

We use both real-world and randomly generated graph test instances. For the real-world graphs, we selected from the University of Florida Sparse Matrix Collection [20], the Pajek dataset [8], Social Graphs in Movies [40], the Network Data Repository [58], and the COLOR 02/03/04 instances [18], a subset of infrastructure, social, and communication networks that have been commonly used for studying critical elements [73]. For the randomly generated graphs, we constructed several random instances applying three traditional random graph generation models. All the graphs range in size from 30 to 232 nodes, are connected (if the original graph is disconnected, we used the largest component), and have different edge densities. The total count of graphs used is then 135 randomly generated graphs plus 16 real-world world graphs, for a total of 151 graphs, as described below.

- **Social networks from popular culture:** `anna-138`, `david-87`, `huck-69`, and `jean-77` are social networks created with the characters from the books *Anna Karenina* by Leo Tolstoy, *David Copperfield* by Charles Dickens, *Adventures of Huckleberry Finn* by Mark Twain, and *Les Misérables* by Victor Hugo, respectively; `forrest-gump-94` and `titanic-70` are social networks created with the characters from these two popular movies; `Lindenstrasse-232` is a social network created describing personal relationships between the characters of a German soap opera of the same name.

- **Social networks from scientific articles:** `sanjuansur-75` is a social network of families in a rural area of Costa Rica; `krebs-62` is the social network of the 9/11 hijackers and their associates; `prison-67` is a social network of prison inmates in the 1950s; `dolphins-62` is a social network of a community of wild dolphins living off Doubtful Sound, New Zealand; `hi-tech-33` is the friendship network of the employees in a small hi-tech company; `karate-34` is the social network of a karate club at a U.S. university in the 1970s.

- **Infrastructure networks:** `miles250-92` is a transportation network composed of cities from the United States connected by the road network of the 1940s (an edge between two cities means that the distance between them is no greater than 250 road miles) and `ieeebus-118` represents the IEEE 118 Bus test case, a power subsystem in the Midwestern U.S. in the 1960s.

- **Communications and collaboration networks:** `ca-sandi-auths-86` is a collaboration network of scientists at the Sandia National Labs.

- **Random Graphs:** In addition to the real world graphs, we also generated several random instances using three well-known random graph generator models: the Erdös-Rényi model [25] to produce uniform random graphs (URG), the Barabási-Albert model [4] to generate scale-free graphs (BA), and the Watts-Strogatz model [77] to produce small-world graphs (WS). We generated instance sets with: (1) 30 nodes and 60, 90, and 180 edges; (2) 40 nodes and 80, 120, and 240 edges; (3) 60 nodes and 120, 180, and 360 edges; (4) 70 nodes and 140, 210, and 420 edges; and (5) 100 nodes and 200, 300, and 600 edges (the actual number of edges for the BA and WS graphs slightly differs from the expected target due to the nature of the generators). For each of the above node/edge configurations, we generated three random samples for each graph type. In total, we have a combination of three types of graphs, five sizes (i.e., values for n), three edge densities (given by the number of edges), and three random samples for a total of $3 \times 5 \times 3 \times 3 = 135$ randomly generated graphs.

8.2 Results for Detecting Critical Cliques

We used the `URG`, `BA`, and `WS` randomly generated graphs to test the performance of the branch-price-and-cut scheme. For each of these 135 graphs, we solve the problem of detecting critical cliques for three different values of the budget. We set the value of b to 1, 3, and 5 for the graphs with 30 and 40 nodes; 3, 5, and 10 for the graphs with 60 and 70 nodes; and 5, 10, and 15 for the graphs with 100 nodes. Tables 6 and 7 of the Appendix Part C report the results obtained when minimizing the CNP and the LC objectives, respectively (also briefly reported for some instances and the CNP objective in Table 1 below). These tables report the average computational time in seconds, the average upper and lower bounds on the optimal solutions, and the average optimality gap (in %) evaluated as $(UB - LB)/LB \times 100$. Furthermore, Table 2 below, as well as Tables 8 and 9 in the Appendix Part C present the average number of times the separation algorithm for the path constraints and the pricing subproblem were called, as well as the average total execution times for both algorithms in seconds. The values in these tables were calculated by averaging the results of the three randomly generated graphs for each of the node/edge configurations. For the convenience of presentation, the values in the tables are rounded to their closest integer values.

Table 1: Computational times, bounds, and gaps for detecting critical cliques for the CNP objective.

			`URG`				`BA`				`WS`			
n	m	b	Time (s)	UB	LB	Gap %	Time (s)	UB	LB	Gap %	Time (s)	UB	LB	Gap %
		1	1	334	334	0	1	220	220	0	0	342	342	0
	60	3	6	95	95	0	1	21	21	0	0	143	143	0
		5	2	14	14	0	0	3	3	0	0	20	20	0
		1	0	325	325	0	0	246	246	0	0	317	317	0
30	90	3	9	153	153	0	7	21	21	0	0	144	144	0
		5	4	24	24	0	8	1	1	0	0	18	18	0
		1	0	276	276	0	0	177	177	0	0	276	276	0
	180	3	36	100	100	0	29	28	28	0	4	91	91	0
		5	16	7	7	0	66	1	1	0	0	1	1	0
		3	394	1,086	1,086	0	272	264	264	0	3	1,091	1,091	0
	120	5	3,785	606	606	0	1,685	85	85	0	7	597	597	0
		10	3,895	32	32	0	216	4	4	0	1	59	53	13
		3	512	1,192	1,192	0	1,139	593	593	0	21	1,113	1,113	0
60	180	5	5,286	836	772	9	6,324	237	237	0	203	667	628	6
		10	7,200	74	52	43	5,685	6	6	0	3	63	30	105
		3	1,235	1,112	1,112	0	2,287	691	691	0	1,048	975	975	0
	360	5	7,200	754	600	26	7,200	269	146	98	7,200	529	315	68
		10	7,200	160	12	1,323	7,200	3	3	0	21	31	1	4,011
		5	7,200	2,936	2,427	21	7,200	1,017	1,017	0	3,060	2,774	2,204	26
	200	10	7,200	1,283	329	283	7,200	57	57	0	484	793	294	172
		15	7,200	109	65	65	7,200	13	13	0	42	134	69	93
		5	7,200	3,376	2,897	17	7,200	1,702	1,546	11	7,084	2,986	2,368	26
100	300	10	7,200	2,396	679	253	7,200	114	114	0	2,192	1,278	256	401
		15	7,200	703	101	590	7,200	22	20	11	3,231	212	42	406
		5	7,200	3,269	2,785	17	7,200	2,302	1,598	45	7,200	2,628	1,971	33
	600	10	7,200	2,324	626	272	7,200	935	134	578	7,200	1,259	122	940
		15	7,200	1,295	59	2,099	7,200	16	11	56	6,869	163	2	9,190

There are several observations that can be obtained from these results. First, regarding the difficulty of the problem, it is clear that finding optimal solutions for larger and denser graphs is significantly more difficult, which is not surprising because the number of paths between every node pair and the total number of cliques increases dramatically with the size of the graphs. In the context of branch-price-and-cut, this translates into more calls to the path constraint separation algorithm and the pricing subproblem, as evidenced by the results of Tables 8 and 9 of the Appendix Part C (briefly in Table 2 below).

With regard to the type of objective and the graph generator model, solving the problem for the CNP objective seems to be slightly easier than for the LC objective, as a few additional instances were solved to optimality. In general, for the LC objective, the proposed algorithm seems to struggle with medium-sized to large instances for all three types of randomly generated graphs, as it seems harder to raise the lower bounds within the time limit for this case. Additionally, the instances generated with the `BA` model are in general easier to solve and, interestingly, become more disconnected with the removal of the critical cliques. This can be partly explained because the preferential attachment mechanism (i.e., the more connected a node is during the generation of the graph the higher are its chances of gaining more connections) used in this model produces hubs that end up being part of the critical node structures. This result goes along with the widely accepted belief [5] that graphs of this nature (i.e., power-law graphs) are often more vulnerable to targeted attacks than graphs whose structure is closer to uniform random ones.

With respect to the value of the budget, larger values for b have a high impact on the optimality gap. This is intuitive because more portions of the graph can be removed if the budget is increased, which may introduce a significant number of alternative fractional solutions.

Table 2: Number of calls within the branch-price-and-cut and overall execution times of the path constraint separation algorithm and the pricing subproblem for the CNP objective.

			Path Constraint Separation						Pricing Subproblem					
			`URG`		`BA`		`WS`		`URG`		`BA`		`WS`	
n	m	b	Calls	Time	Calls	Time	Calls	Time	Calls	Time	Calls	Time	Calls	Time
		1	136	0	61	0	160	0	173	0	72	0	206	0
	60	3	446	0	72	0	1,607	0	586	0	94	0	2,148	0
		5	420	0	45	0	2,533	1	578	0	60	0	3,393	0
		1	68	0	54	0	79	0	88	0	76	0	108	0
30	90	3	1,618	0	128	0	2,182	0	2,125	0	178	0	2,883	0
		5	4,801	1	11	0	4,132	1	6,319	0	20	0	5,393	0
		1	13	0	24	0	30	0	49	0	53	0	93	0
	180	3	4,860	1	1,812	0	6,761	1	6,579	0	2,847	0	9,142	0
		5	30,828	6	89	0	5,025	1	46,713	3	235	0	10,116	1
		3	6,545	11	335	0	9,861	17	8,810	1	441	0	13,149	1
	120	5	44,658	71	1,186	2	111,358	177	59,478	5	1,566	0	148,685	14
		10	66,770	87	181	0	912,459	1,182	90,202	8	315	0	1,212,766	461
		3	23,465	40	716	1	11,682	19	34,468	3	964	0	15,586	1
60	180	5	140,067	221	9,739	15	116,211	179	185,366	21	12,927	1	153,983	15
		10	676,650	929	1,154	1	1,082,407	1,452	881,725	238	1,696	0	1,362,241	550
		3	29,156	45	25,156	38	19,269	29	45,302	5	35,110	4	26,535	3
	360	5	168,989	252	262,045	382	153,087	221	225,376	29	323,123	97	187,168	27
		10	357,271	512	3,289	4	824,580	1,111	422,623	125	6,646	1	991,801	1,093
		5	30,925	220	36,474	252	23,840	271	39,788	8	48,464	8	30,894	8
	200	10	73,113	515	56,626	323	84,868	544	87,838	20	75,330	12	102,307	24
		15	641,840	3,744	5,662	32	638,130	3,649	823,320	279	8,943	1	818,196	239
		5	23,080	165	38,566	269	29,285	204	29,684	7	51,220	9	37,673	8
100	300	10	14,772	106	144,586	834	47,498	308	17,701	3	190,568	31	56,909	12
		15	90,446	579	385,167	2,174	334,054	1,963	108,027	24	536,125	161	410,229	147
		5	15,343	109	41,987	283	34,315	231	20,484	5	53,469	14	45,144	11
	600	10	15,679	104	86,346	539	39,016	240	19,294	4	103,985	29	46,351	11
		15	39,285	246	486,936	2,799	245,187	1,475	45,428	10	664,238	588	297,161	110

Second, based on the results from Tables 8 and 9 in the Appendix C, it is evident that the total running times of both the path constraint separation algorithm and the pricing subproblem are not bottlenecks of the overall execution—despite the large number of calls. This is easy to explain for the path constraint separation algorithm because solving APSP problems for graphs of 100 nodes is performed quite quickly. On the other hand, the short execution times for the pricing subproblem can be explained by the preprocessing algorithm. Based on our experiments, we have observed that such an algorithm plays an important role in reducing the computational time of the subproblem, particularly in latter instances of the branch-price-and-cut where a lesser number of good candidate cliques are yet to be generated thus producing an efficient optimality certificate of the subproblem.

Third, we have observed that most of the instances in which our approach struggles to efficiently close the optimality gap correspond to cases for which the default heuristics of the optimizer (e.g., variable rounding heuristics) fail to identify good feasible solutions in the early stages of the branch-and-bound tree. This generally occurs when solving the problem over larger and denser graphs because the algorithm requires first separating many more path constraints before finding potential integer solutions that are actually feasible. We believe that incorporating a heuristic (even simple greedy heuristics like the ones depicted in Figure 2) to produce a warm-start could be quite fruitful; particularly for tackling instances with the LC objective which, as mentioned before, pose a more difficult challenge to our approach.

Fourth, the experiments provide evidence that the proposed framework can be used to efficiently solve the problem of detecting critical node structures. We find particularly interesting the fact that our framework can tackle a wide variety of critical node structure detection problems by simply adapting the pricing subproblem. It is important to mention, though, that the efficiency of the algorithm is highly correlated with the complexity of the pricing subproblem. Therefore, for detecting critical node structures for which (36) is difficult to compute, the efficacy of the proposed approach could be limited. In such cases, other approaches may potentially yield better results.

8.3 Results for Detecting Critical Stars

We used the real-world graphs to compare the performance of the proposed framework when using both the regular path constraints and the $\mathcal{T}$-path constraints, as well as to analyze the impact that the 3-clique and 4-cycle inequalities may have over the solution quality. For each of these 16 graphs, we solve the problem of detecting critical stars for three different values of the budget. Furthermore, noting that the difference in the complexity of detecting critical cliques between both the CNP and LC objectives is relatively mild, we focus our attention on the CNP.

Tables 3 (briefly) below and 10 (fully) in the Appendix Part C report the optimal solutions of the linear relaxations obtained with the original path constraints (path), the $\mathcal{T}$-path constraints ($\mathcal{T}$-path), and the $\mathcal{T}$-path constraints strengthened with the 3-clique and 4-cycle inequalities ($\mathcal{T}$-p-KC). As expected, the $\mathcal{T}$-path inequalities yielded a tighter linear relaxation in 36 out of the 48 runs, with an average improvement of 6.55% and a maximum of 27.38%. In contrast, the improvement produced by adding the 3-clique and 4-cycle constraints is rather mild, being close to 0.06% on average and producing a maximum of 1.39%.

The relative minor impact of the 3-clique and 4-cycle constraints can be partially explained by the difference between the number of small cliques and chordless cycles versus the number of path (or $\mathcal{T}$-path) inequalities added during the execution of the algorithm. Given the low density of the graphs, the average number of 3-clique and 4-cycle inequalities is merely 335 and 47, respectively, whereas the number of generated path (or $\mathcal{T}$-path) inequalities is in most cases of the order of thousands (see Table 12 in the Appendix Part C).

In terms of the overall performance, Tables 4 (below) and 11 (in the Appendix Part C) report

Table 3: Optimal solutions of the linear relaxations for detecting critical stars for the CNP objective. The values are the results obtained with the original path constraints, the $\mathcal{T}$-path constraints, and the $\mathcal{T}$-path constraints strengthened with the 3-clique and 4-cycle inequalities. The ratios between such results are also presented.

Name	n	m	b	path	$\mathcal{T}$-path	$\mathcal{T}$-p-KC	$\frac{(\mathcal{T}\text{-path})-\text{path}}{\text{path}}\%$	$\frac{(\mathcal{T}\text{-p-KC})-(\mathcal{T}\text{-path})}{\mathcal{T}\text{-path}}\%$
`high-tech-33`	33	91	250	303.083	322.440	323.101	6.39	0.205
`high-tech-33`	33	91	350	224.025	247.000	247.571	10.26	0.231
`high-tech-33`	33	91	590	72.700	87.935	89.156	20.96	1.389
`prison-67`	67	142	350	1,303.145	1,387.840	1,387.840	6.50	0.000
`prison-67`	67	142	590	715.198	861.962	861.962	20.52	0.000
`prison-67`	67	142	1,250	103.154	123.828	123.828	20.04	0.000
`sanjuansur-75`	75	144	350	1,651.034	1,745.650	1,745.650	5.73	0.000
`sanjuansur-75`	75	144	590	901.822	1,054.760	1,054.760	16.96	0.000
`sanjuansur-75`	75	144	1,200	155.014	193.556	193.556	24.86	0.000
`forest-gump-94`	94	271	590	114.960	115.800	115.800	0.73	0.000
`forest-gump-94`	94	271	1,200	60.400	61.489	61.489	1.80	0.000
`forest-gump-94`	94	271	1,750	34.000	38.697	38.697	13.81	0.000
`LindenStrasse-232`	232	303	1,200	1,975.530	2,472.000	2,474.500	25.13	0.101
`LindenStrasse-232`	232	303	1,750	737.526	796.500	796.500	8.00	0.000
`LindenStrasse-232`	232	303	2,400	346.722	383.103	383.274	10.49	0.045

the upper and lower bounds on the optimal solutions, as well as the optimality gaps for the cases described above. Additionally, Table 12 in the Appendix Part C reports the execution time in seconds, the number of times the separation routines are called, and the total time used for separating each version of the path constraints in seconds. For each of the instances, we indicate in bold the best values obtained, i.e., the smallest UB, the largest LB, and the smaller optimality gap; and the shortest solution time, the least number of separation calls, and the shortest overall separation time, unless the results are the same for all three cases.

Table 4: Bounds and optimality gaps for detecting critical stars for the CNP objective.

		UB			LB			Gap %		
Name	b	path	$\mathcal{T}$-path	$\mathcal{T}$-p-KC	path	$\mathcal{T}$-path	$\mathcal{T}$-p-KC	path	$\mathcal{T}$-path	$\mathcal{T}$-p-KC
`high-tech-33`	250	351	351	351	351	351	351	0	0	0
`high-tech-33`	350	276	276	276	276	276	276	0	0	0
`high-tech-33`	590	162	162	162	162	162	162	0	0	0
`prison-67`	350	1,498	1,498	1,498	1,498	1,498	1,498	0	0	0
`prison-67`	590	1,109	1,109	1,109	**1,109**	1,050	**1,109**	**0**	6	**0**
`prison-67`	1,250	159	159	159	159	159	159	0	0	0
`sanjuansur-75`	350	1,968	1,968	1,968	1,968	1,968	1,968	0	0	0
`sanjuansur-75`	590	1,433	1,433	1,433	**1,433**	1,391	1,378	**0**	3	4
`sanjuansur-75`	1,200	206	206	206	206	206	206	0	0	0
`forest-gump-94`	590	135	135	135	**135**	123	122	**0**	10	10
`forest-gump-94`	1,200	66	66	66	66	66	66	0	0	0
`forest-gump-94`	1,750	**45**	46	**45**	**40**	41	**40**	14	**12**	**12**
`LindenStrasse-232`	1,200	10,016	5,197	**4,345**	2,338	2,616	**2,633**	328	99	**65**
`LindenStrasse-232`	1,750	830	830	830	830	830	830	0	0	0
`LindenStrasse-232`	2,400	405	403	403	374	**403**	**403**	8	**0**	**0**

Interestingly, in spite of producing a worse linear relaxation, the formulation with the original path inequalities often produced a better overall lower bound and better solution times in comparison with the formulations that used the $\mathcal{T}$-path constraints. The reasons that explain this behavior are twofold. First, the $\mathcal{T}$-path constraints are significantly denser, therefore, performing each iteration of the dual simplex (the LP solver used) generally takes longer. Second, the separation procedure for the $\mathcal{T}$-path constraints is more complex and thus requires further iterations to be completed. Clearly, the number of $\mathbf{x}$ variables in the original path constraints is given by the

Table 5: Execution times for detecting critical stars for the CNP objective.

		Solution Time (s)			Separation Calls			Total Separation Time (s)		
Name	b	path	$\mathcal{T}$-path	$\mathcal{T}$-p-KC	path	$\mathcal{T}$-path	$\mathcal{T}$-p-KC	path	$\mathcal{T}$-path	$\mathcal{T}$-p-KC
`high-tech-33`	250	**18**	21	19	1,238	1,050	**913**	**1**	4	3
`high-tech-33`	350	17	**11**	15	902	**525**	618	**1**	2	2
`high-tech-33`	590	**583**	3,720	**3,972**	124,579	527,994	567,858	**67**	2,385	2,515
`prison-67`	350	272	325	**147**	3,209	4,259	**1,597**	**15**	83	31
`prison-67`	590	**4,130**	7,200	6,662	**88,773**	178,394	155,216	**396**	3,786	3,289
`prison-67`	1,250	2,467	**476**	602	180,829	**39,103**	52,300	675	**297**	402
`sanjuansur-75`	350	464	415	**336**	4,748	**3,129**	3,416	**32**	83	86
`sanjuansur-75`	590	5,133	7,200	7,200	**101,947**	144,016	143,676	**640**	3,926	3,974
`sanjuansur-75`	1,200	331	**5**	13	16,020	182	758	87	1	6
`forest-gump-94`	590	**136**	7,200	7,200	7,940	39,044	44,069	**73**	6,306	6,316
`forest-gump-94`	1,200	**1,077**	2,043	2,480	75,069	**45,960**	56,804	679	**214**	265
`forest-gump-94`	1,750	7,200	7,200	7,200	552,366	149,815	**98,414**	4,994	700	**457**
`LindenStrasse-232`	1,200	7,200	7,200	7,200	**9,926**	12,969	15,470	**1,729**	5,014	5,065
`LindenStrasse-232`	1,750	1,457	**569**	**643**	6,293	4,093	4,744	877	**387**	437
`LindenStrasse-232`	2,400	7,200	**4,213**	4,432	42,779	43,640	**41,725**	5,801	**3,188**	3,472

length of the corresponding path, which is $O(n)$, whereas the number of $\mathbf{z}$ variables in the $\mathcal{T}$-path constraints depends on the number of node structures that contain nodes in the corresponding path, which is $O(|\mathcal{T}|)$.

The results of Table 5 provide evidence that the separation process has indeed a strong impact on the overall execution time. In fact, it is observed that separating the $\mathcal{T}$-path constraints takes on average more than four times the time required to separate the original path constraints. In other words, the advantage gained by the $\mathcal{T}$-path constraints of producing a tighter linear relaxation is greatly attenuated by the extra time required by the separation routine.

Finally, the results obtained with and without the 3-clique and 4-cycle inequalities (i.e., the $\mathcal{T}$-p-KC and $\mathcal{T}$-path cases) indicate that the differences between both cases are not produced by such inequalities, but may correspond to random decisions made by the optimizer. The slim difference in the linear relaxation bounds contrasted with the overall solution times reveal that what really determines the overall execution times is not the quality of the relaxation, but the number of times the separation procedure is called.

9 Concluding Remarks

In this paper we proposed a generic mathematical programming approach for optimally deteriorating two connectivity properties of a graph by removing a collection of critical node structures, subject to a budgetary constraint. The proposed approach generalizes other existing models whose scope is restricted to removing individual and unrelated nodes by providing a flexible methodological tool that can be used detect a wide variety of critical node structures. More specifically, our framework consists of a branch-price-and-cut approach that includes a particular branching strategy, a generic pricing scheme, and several preprocessing algorithms used to speed up the generation of new columns.

The proposed approach also introduces a new set of valid inequalities that can be used to strengthen our formulations. We illustrated how to use the proposed framework by solving the cases where the structures form stars or cliques, and provided general directions on how to adapt the framework for solving the problem for other types of critical node structures as well. Our extensive computational experiments tested the proposed framework under various settings and reveal several interesting insights. In particular, the results showed that standard MIP solvers in conjunction with our approach can tackle relatively large instances in a reasonable amount of time. The computational results also revealed an underlying trade-off between the solution quality of the linear relaxation of the given formulations and the solution times required to solve them.

For applications that require the branch-price-and-cut approach, we observed that the efficiency of our algorithm is highly correlated with the complexity of the pricing subproblem. Therefore, the efficacy of the proposed approach could be hindered when detecting critical node structures for which pricing new candidates poses a difficult challenge.

As far as future work is concerned, the further development of valid inequalities and other general tools to better handle the large number of generated constraints, as well as the design of approximate and heuristic solution methods, should be a fruitful research direction. Furthermore, it may be interesting to test the proposed framework for detecting other types of critical node structures.

References

[1] B. Addis, M.D. Summa, and A. Grosso, Identifying critical nodes in undirected graphs: Complexity results and polynomial algorithms for the case of bounded treewidth, Discr Appl Math 161 (2013), 2349–2360.

[2] Y. Agarwal, K. Mathur, and H.M. Salkin, A set-partitioning-based exact algorithm for the vehicle routing problem, Networks 19 (1989), 731–749.

[3] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin, Network flows: Theory, algorithms, and applications, Prentice Hall, New Jersey, USA, 1993.

[4] R. Albert and A.L. Barabási, Statistical mechanics of complex networks, Reviews Modern Phys 74 (2002), 47.

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

[6] A. Arulselvan, C.W. Commander, L. Elefteriadou, and P.M. Pardalos, Detecting critical nodes in sparse graphs, Comput Oper Res 36 (2009), 2193–2200.

[7] C. Barnhart, E.L. Johnson, G.L. Nemhauser, M.W.P. Savelsbergh, and P.H. Vance, Branch-and-price: Column generation for solving huge integer programs, Oper Res 46 (1998), 316–329.

[8] V. Batagelj and A. Mrvar, Pajek datasets, http://vlado.fmf.uni-lj.si/pub/networks/data/, (Accessed August, 2017).

[9] C. Bazgan, S. Toubaline, and Z. Tuza, The most vital nodes with respect to independent set and vertex cover, Discr Appl Math 159 (2011), 1933–1946.

[10] C. Bazgan, S. Toubaline, and D. Vanderpooten, “Complexity of determining the most vital elements for the 1-median and 1-center location problems,” Combinatorial optimization and applications, W. Wu and O. Daescu (Editors), Springer Berlin Heidelberg, 2010, Vol. 6508, pp. 237–251.

[11] T. Berthold, G. Gamrath, A.M. Gleixner, S. Heinz, T. Koch, and Y. Shinano, Solving mixed integer linear and nonlinear problems using the SCIP Optimization Suite, Zib-report 12–17, Zuse Institute Berlin, Takustr. 7, 14195 Berlin, 2012.

[12] R.E. Bixby, J.W. Gregory, I.J. Lustig, R.E. Marsten, and D.F. Shanno, Very large-scale linear programming: A case study in combining interior point and simplex methods, Oper Res 40 (1992), 885–897.

[13] H. Bodlaender, A. Grigoriev, N.V. Grigorieva, and A. Hendriks, "The valve location problem in simple network topologies," Graph-theoretic concepts in computer science, H. Broersma, T. Erlebach, T. Friedetzky, and D. Paulusma (Editors), Springer Berlin Heidelberg, 2008, Vol. 5344, pp. 55–65.

[14] S.P. Borgatti, Identifying sets of key players in a social network, Comput Math Organization Theory 12 (2006), 21–34.

[15] J.M. Bourjolly, G. Laporte, and G. Pesant, An exact algorithm for the maximum k-club problem in an undirected graph, Eur J Oper Res 138 (2002), 21 – 28.

[16] T. Christof, A. Löbel, and M. Stoer, PORTA: A POlyhedron Representation Transformation Algorithm, http://porta.zib.de/, (Accessed August, 2017).

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

[18] COLOR 02/03/04, Graph coloring and its generalizations, http://mat.gsia.cmu.edu/COLOR03/ (Accessed August, 2017).

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

[20] T.A. Davis and Y. Hu, The University of Florida Sparse Matrix Collection, ACM Trans Math Software 38 (2011), 1–25.

[21] M. Di Summa, A. Grosso, and M. Locatelli, Complexity of the critical node problem over trees, Comput Oper Res 38 (2011), 1766–1774.

[22] M. Di Summa, A. Grosso, and M. Locatelli, Branch and cut algorithms for detecting critical nodes in undirected graphs, Comput Optim Appl 53 (2012), 649–680.

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

[24] O. du Merle, D. Villeneuve, J. Desrosiers, and P. Hansen, Stabilized column generation, Discr Math 194 (1999), 229 – 237.

[25] P. Erdös and A. Rényi, On random graphs, I, Publicationes Mathematicae (Debrecen) 6 (1959), 290–297.

[26] M.G. Everett and S.P. Borgatti, The centrality of groups and classes, J Math Sociology 23 (1999), 181–201.

[27] D. Feillet, A tutorial on column generation and branch-and-price for vehicle routing problems, 4OR 8 (2010), 407–424.

[28] G.N. Frederickson and R. Solis-Oba, Increasing the weight of minimum spanning trees, J Algorithms 33 (1999), 244–266.

[29] M.R. Garey and D.S. Johnson, Computers and intractability: A guide to the theory of NP-completeness, W. H. Freeman & Co., New York, NY, USA, 1979.

[30] D. Granata, G. Steeger, and S. Rebennack, Network interdiction via a critical disruption path: Branch-and-price algorithms, Comput Oper Res 40 (2013), 2689 – 2702.

[31] M. Grötschel and Y. Wakabayashi, Facets of the clique partitioning polytope, Math Program 47 (1990), 367–387.

[32] T.H. Grubesic, T.C. Matisziw, A.T. Murray, and D. Snediker, Comparative approaches for assessing network vulnerability, Int Regional Sci Review 31 (2008), 88–112.

[33] T.H. Grubesic and A.T. Murray, Vital nodes, interconnected infrastructures, and the geographies of network survivability, Ann Assoc Am Geogr 96 (2006), 64–83.

[34] P. Hansen and B. Jaumard, Cluster analysis and mathematical programming, Math Program 79 (1997), 191–215.

[35] R. Hewett, Toward identification of key breakers in social cyber-physical networks, Syst, Man, Cybernetics (SMC), 2011 IEEE Int Conference , Oct 2011, pp. 2731–2736.

[36] J. Hopcroft and R. Tarjan, Algorithm 447: Efficient algorithms for graph manipulation, Commun ACM 16 (1973), 372–378.

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

[38] E. Israeli and R.K. Wood, Shortest-path network interdiction, Networks 40 (2002), 97–111.

[39] E. Jenelius, T. Petersen, and L.G. Mattsson, Importance and exposure in road network vulnerability analysis, Transp Res Part A: Pol Practice 40 (2006), 537–560.

[40] J. Kaminski, M. Schober, R. Albaladejo, O. Zastupailo, and C. Hidalgo, Moviegalaxies - social networks in movies, http://moviegalaxies.com/, (Accessed August, 2017).

[41] K.T. Kennedy, R.F. Deckro, J.T. Moore, and K.M. Hopkinson, Nodal interdiction, Math Comput Modelling 54 (2011), 3116 – 3125.

[42] M. Lalou, M.A. Tahraoui, and H. Kheddouci, The critical node detection problem in networks: A survey, Comput Sci Review 28 (2018), 92 – 117.

[43] J.M. Lewis and M. Yannakakis, The node-deletion problem for hereditary properties is NP-complete, J Comput System Sci 20 (1980), 219–230.

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

[45] M.E. Lübbecke and J. Desrosiers, Selected topics in column generation, Oper Res 53 (2005), 1007–1023.

[46] F. Mahdavi Pajouh, V. Boginski, and E.L. Pasiliao, Minimum vertex blocker clique problem, Networks 64 (2014), 48–64.

[47] F. Mahdavi Pajouh, J.L. Walteros, V. Boginski, and E.L. Pasiliao, Minimum edge blocker dominating set problem, Eur J Oper Res 247 (2015), 16 – 26.

[48] R.E. Marsten, W.W. Hogan, and J.W. Blankenship, The boxstep method for large-scale optimization, Oper Res 23 (1975), 389–405.

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

[50] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon, Network motifs: Simple building blocks of complex networks, Sci 298 (2002), 824–827.

[51] Y.S. Myung and H.J. Kim, A cutting plane algorithm for computing k-edge survivability of a network, Eur J Oper Res 156 (2004), 579–589.

[52] J. Naoum-Sawaya and C. Buchheim, Robust critical node selection by Benders decomposition, INFORMS J Comput 28 (2016), 162–174.

[53] G.L. Nemhauser and W.B. Widhelm, A modified linear program for columnar methods in mathematical programming, Oper Res 19 (1971), 1051–1060.

[54] G.L. Nemhauser and L.A. Wolsey, Integer and combinatorial optimization, Wiley-Interscience, New York, NY, USA, 1988.

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

[56] D. Ortiz-Arroyo and D.M. Hussain, An information theory approach to identify sets of key players, Proc 1st Eur Conference Intelligence Security Informatics, Springer-Verlag, 2008, pp. 15–26.

[57] P.R.J. Östergård, A new algorithm for the maximum-weight clique problem, Nordic J Comput 8 (2001), 424–436.

[58] R.A. Rossi and N.K. Ahmed, The network data repository with interactive graph analytics and visualization, Proc Twenty-Ninth AAAI Conference Artificial Intelligence, 2015, pp. 4292–4293, `http://networkrepository.com` (Accessed August, 2017).

[59] L.M. Rousseau, M. Gendreau, and D. Feillet, Interior point stabilization for column generation, Oper Res Lett 35 (2007), 660–668.

[60] D.M. Ryan and B.A. Foster, "An integer programming approach to scheduling," Computer scheduling of public transport urban passenger vehicle and crew scheduling, A. Wren (Editor), North-Holland, 1981, pp. 269–280.

[61] J. Salmeron, K.R. Wood, and R. Baldick, Analysis of electric grid security under terrorist threat, IEEE Trans Power Syst 19 (2004), 905–912.

[62] D.F. Shanno and R.L. Weil, Technical note—"Linear" programming with absolute-value functionals, Oper Res 19 (1971), 120–124.

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

[64] S. Shen, J.C. Smith, and R. Goli, Exact interdiction models and algorithms for disconnecting networks via node deletions, Discr Optim 9 (2012), 172 – 188.

[65] Y. Shen, N.P. Nguyen, Y. Xuan, and M.T. Thai, On the discovery of critical links and nodes for assessing network vulnerability, IEEE/ACM Trans Networking 21 (2013), 963–973.

[66] V. Spirin and L.A. Mirny, Protein complexes and functional modules in molecular networks, Proc National Acad Sci 100 (2003), 12123–12128.

[67] Z. Tao, F. Zhongqian, and W. Binghong, Epidemic dynamics on complex networks, Progress Natural Sci 16 (2005), 452–457.

[68] R. van der Zwaan, A. Berger, and A. Grigoriev, "How to cut a graph into many pieces," Theory and applications of models of computation, M. Ogihara and J. Tarui (Editors), Springer Berlin Heidelberg, 2011, Vol. 6648 of Lecture Notes in Computer Science, pp. 184–194.

[69] M. Ventresca, Global search algorithms using a combinatorial unranking-based problem representation for the critical node detection problem, Comput Oper Res 39 (2012), 2763 – 2775.

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

[71] A. Veremyev and V. Boginski, Identifying large robust network clusters via new compact formulations of maximum k-club problems, Eur J Oper Res 218 (2012), 316 – 326.

[72] A. Veremyev, V. Boginski, and E.L. Pasiliao, Exact identification of critical nodes in sparse networks via new compact formulations, Optim Lett 8 (2014), 1245–1259.

[73] A. Veremyev, O.A. Prokopyev, and E.L. Pasiliao, An integer programming framework for critical elements detection in graphs, J Combinatorial Optim 28 (2014), 233–273.

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

[75] J.L. Walteros and P.M. Pardalos, "A decomposition approach for solving critical clique detection problems," Experimental algorithms, R. Klasing (Editor), Springer Berlin Heidelberg, 2012, Vol. 7276 of Lecture Notes in Computer Science, pp. 393–404.

[76] J.L. Walteros, C. Vogiatzis, E.L. Pasiliao, and P.M. Pardalos, Integer programming models for the multidimensional assignment problem with star costs, Eur J Oper Res 235 (2014), 553 – 568.

[77] D.J. Watts and S.H. Strogatz, Collective dynamics of small-world networks, Nature 393 (1998), 440–442.

[78] R. Wollmer, Removing arcs from a network, Oper Res 12 (1964), 934–940.

[79] R.K. Wood, Deterministic network interdiction, Math Comput Modeling 17 (1993), 1–18.

[80] R. Zenklusen, Matching interdiction, Discr Appl Math 158 (2010), 1676–1690.

[81] R. Zenklusen, B. Ries, C. Picouleau, D. de Werra, M.C. Costa, and C. Bentz, Blockers and transversals, Discr Math 309 (2009), 4306–4314.

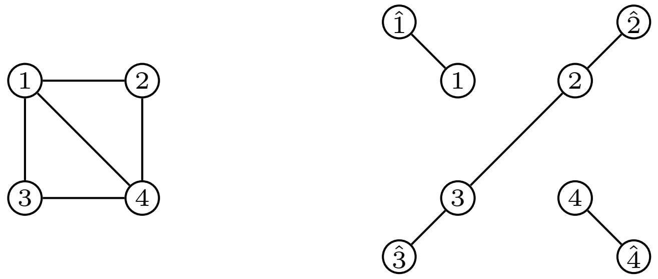
Appendix

A Mathematical Proofs

Theorem 1 *CCP for both the CNP and LC measures is strongly NP-complete.*

Proof of Theorem 1. We first show the intractability of CCP for the CNP metric via a polynomial-time reduction from k-COLORABILITY, whose decision version is stated as follows: given a simple non-empty graph $G = (V, E)$ and an integer $k > 0$, is it possible to color the nodes in V with at most k colors so that no pair of adjacent nodes share the same color. Given the graph G from any k-colorability instance, we can construct the following graph $\hat{G}$ in polynomial time. First, we start with the complement of G. That is, the graph given by node set V and edge set $\binom{V}{2} \setminus E$. Then, for each node $i \in V$, we create a copy denoted by $\hat{i}$ and add an edge connecting i with $\hat{i}$. Let $\hat{V}$ and $\hat{E}$ be the set of node copies and the set of these additional edges, respectively. The resulting graph is thus $\hat{G} = (V \cup \hat{V}, (\binom{V}{2} \setminus E) \cup \hat{E})$. Figure 4 provides an example of a 4-node graph G and the corresponding transformation into $\hat{G}$. We argue that graph G is k-colorable if and only if there is a collection $\mathcal{K}$ of at most $b = k$ cliques so that the total number of connected pairs in $\hat{G}[V \setminus V(\mathcal{K})]$ is $r = 0$.

Figure 4: Graph transformation for reducing k-colorability to CCP



(a) A 4-node graph G from a k-colorability instance.

(b) The transformed graph $\hat{G}$.

To prove necessity, assume there exists a set $\mathcal{K}$ of at most $b = k$ cliques such that the total number of connected node pairs in $\hat{G}[V \setminus V(\mathcal{K})]$ is zero. We first show that without loss of generality we can assume the cliques in $\mathcal{K}$ are disjoint.

Claim 1 *Any solution $\mathcal{K}$ of CCP for CNP with cliques that overlap (i.e., a solution $\mathcal{K}$ in which there exist at least a pair of cliques $T_1, T_2 \in \mathcal{K}$ so that $V(T_1) \cap V(T_2) \neq \emptyset$) can be transformed into a valid solution $\mathcal{K}'$ with at most b disjoint cliques. That is, for all $T_1, T_2 \in \mathcal{K}', V(T_1) \cap V(T_2) = \emptyset$.*

Proof. The set of cliques of a graph is closed under inclusion (i.e., any subset of a clique is also a clique). Therefore, if two or more cliques in $\mathcal{K}$ share some nodes, such nodes can be arbitrarily assigned to one of the cliques that contain them and removed from the others. Consequently, if a clique $T_1 \in \mathcal{T}$ is a subset of a clique $T_2 \in \mathcal{K}$, clique T_1 can be omitted in $\mathcal{K}'$. The result would be a set $\mathcal{K}'$ containing at the most b cliques. □

We can now safely assume the cliques in $\mathcal{K}$ are disjoint. The cliques of set $\mathcal{K}$ can be classified into three groups (two of those possibly empty). First, the set of cliques, say $\mathcal{K}_1$, that contain only nodes from V; second, the set of cliques $\mathcal{K}_2$ that are given by a node in V and its corresponding copy in $\hat{V}$; and third, the set of cliques $\mathcal{K}_3$ that are singleton nodes from $\hat{V}$.

Claim 2 *Any solution $\mathcal{K}$ of CCP for CNP with $\mathcal{K}_1 \subset \mathcal{K}$ can be transformed into a valid solution $\mathcal{K}'$ for which $\mathcal{K}'_2 = \mathcal{K}'_3 = \emptyset$ and $|\mathcal{K}'| \leq |K|$.*

Proof. Any clique $T \in \mathcal{K}_2$ is induced by a node $i \in V$ and its corresponding copy $\hat{i} \in \hat{V}$. Such a clique can be replaced with the clique given by i and the resulting graph will remain fully disconnected. Therefore, we can create a valid solution $\mathcal{K}'$ of the same size as $\mathcal{K}$ with $\mathcal{K}'_2 = \emptyset$. Furthermore, assume that node $\hat{i}$ is one of the cliques in $\mathcal{K}_3$. First, suppose that the corresponding copy $i \in V$ of $\hat{i}$ is not part of any clique in $\mathcal{K}_1$. Note that to disconnect i from $\hat{i}$ we can replace singleton clique $\hat{i}$ with the singleton clique i and the resulting solution is also valid. Second, suppose that i already belongs to a clique in $T \in \mathcal{K}_1$. Then, a solution that does not remove singleton clique $\hat{i}$ is still valid as $\hat{i}$ will have no neighbors in the residual graph. Hence, we can also create a valid solution K' of size at the most equal to $|\mathcal{K}|$ with $\mathcal{K}'_3 = \emptyset$. □

We will now assume without loss of generality that solution $\mathcal{K}$ contains only cliques with nodes from V. Given such a solution $\mathcal{K}$, consider the following way of coloring the nodes of G. For each clique $T \in \mathcal{K}$, we will give all nodes in $V(T)$ the same color. Notice that any clique in $T \in \mathcal{K}$ induces an independent set in G (i.e., its nodes are non-adjacent in G). Therefore, the nodes in $V(T)$ can have the same color. Furthermore, since $|\mathcal{K}| \leq b = k$, the resulting color assignment is valid.

To prove sufficiency, assume that graph G is k-colorable. All nodes of the same color induce an independent set in G and each independent set in G induces a clique in $\hat{G}$. Let $\mathcal{K}$ be the set of those $b = k$ cliques. Notice that removing the cliques in $\mathcal{K}$ from $\hat{G}$ leaves the set of isolated nodes $\hat{V}$, which implies that the total number of connected node pairs left is zero. Since k-colorability is known to be NP-complete [29], CCP is NP-complete as well.

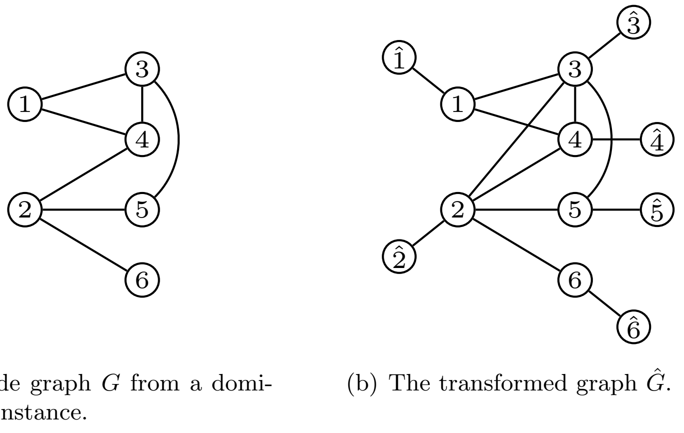
Finally, to show that the decision version of CCP is also NP-complete for the LC connectivity measure, it suffices to notice that the size of the largest component of a graph is at the most one if such a graph is completely disconnected. Therefore, the same reduction can be used to transform an instance of k-colorability, given by a graph G, into an instance of CCP for the LC connectivity measure with $b = k$ and $r = 1$ over the transformed graph $\hat{G}$. □

Theorem 2 *CSP for both the CNP and LC measures is strongly NP-complete*

Proof of Theorem 2. To prove the complexity result for the CNP measure, consider the following polynomial-time reduction from Dominating Set. Given a simple non-empty graph $G = (V, E)$ and an integer $k > 0$, the decision version of the dominating set problem asks if there is a set $N \subseteq V$ of at most k nodes such that any node in V is either in N or is adjacent to a node in N. Given the graph G from any dominating set instance, we can construct a graph $\hat{G}$ by adding to G a copy of each node $i \in V$, denoted $\hat{i}$, and an edge connecting i with $\hat{i}$. Let $\hat{V}$ and $\hat{E}$ be the set of node copies and the set of additional edges, respectively. The resulting graph is then $\hat{G} = (V \cup \hat{V}, E \cup \hat{E})$. Figure 5 provides an example of a 6-node graph G and the corresponding transformation into $\hat{G}$. We argue that graph G has a dominating set of size k if and only if there is a collection $\mathcal{K}$ of at most $b = k$ disjoint stars so that the total number of connected pairs in $\hat{G}[V \setminus V(\mathcal{K})]$ is $r = 0$.

To prove necessity, assume there exists a set $\mathcal{K}$ of at most $b = k$ stars such that the total number of connected node pairs in $\hat{G}[V \setminus V(\mathcal{K})]$ is zero. Similarly, as for the clique case, we first show that without loss of generality we can assume the stars in $\mathcal{K}$ are disjoint.

Figure 5: Graph transformation for reducing dominating set to CNP for CSP



(a) A 6-node graph G from a dominating set instance. (b) The transformed graph $\hat{G}$.

Claim 3 *Any solution $\mathcal{K}$ of CCP for CNP with stars that overlap (i.e., a solution $\mathcal{K}$ in which there exist at least a pair of stars $T_1, T_2 \in \mathcal{K}$ so that $V(T_1) \cap V(T_2) \neq \emptyset$) can be transformed into a valid solution $\mathcal{K}'$ with at most b disjoint stars. That is, for all $T_1, T_2 \in \mathcal{K}', V(T_1) \cap V(T_2) = \emptyset$.*

Proof. First, if say l stars $T_1, T_2, \ldots, T_l$ in $\mathcal{K}$ share the same hub, we can replace them in $\mathcal{K}'$ with a new star $T = \bigcup_{k=1}^{l} T_k$. Second, if the hub $h(T_1)$ of a star $T_1 \in \mathcal{K}$ is the leaf of a star $T_2 \in \mathcal{K}$, star T_2 is replaced in $\mathcal{K}'$ by the star T_2' that results from removing $h(T_1)$ from T_2. Third, if two or more stars in $\mathcal{K}$ share some leaves, as for the cliques, such nodes can be arbitrarily assigned to one of the stars that contain them and removed from the others. The result would be a set $\mathcal{K}'$ containing at the most b stars. □

We can now safely assume the stars in $\mathcal{K}$ are disjoint. The stars of set $\mathcal{K}$ can be classified into two groups (one possibly empty): the set of stars $\mathcal{K}_1$ whose hub $h(T)$ for $T \in \mathcal{K}_1$ is in V, and the set of stars $\mathcal{K}_2$ whose hub $h(T)$ for $T \in \mathcal{K}_2$ is in $\hat{V}$.

Claim 4 *Any solution $\mathcal{K}$ of CNP for CSP can be transformed into a valid solution $\mathcal{K}'$ for which $\mathcal{K}_2' = \emptyset$.*

Proof. If $\mathcal{K}_2 = \emptyset$, the result follows immediately by fixing $\mathcal{K}' = \mathcal{K}$. For the case in which $|\mathcal{K}_2| > 0$, it is easy to see that a star $T \in \mathcal{K}_2$ is either a leafless star given by hub $\hat{i} = h(T) \in V'$ or a star composed of $\hat{i} = h(T) \in \hat{V}$ and its corresponding copy $i \in V$. A valid solution $\mathcal{K}'$ can then be constructed from $\mathcal{K}$ by replacing each star $T \in \mathcal{K}_2$, having a hub $h(T) = \hat{i} \in \hat{V}$, by a star whose hub is the corresponding node i in V. □

Claim 5 *Given a solution $\mathcal{K}$ of CSP for CNP with $\mathcal{K}_2 = \emptyset$, after removing $V(\mathcal{K})$ from $\hat{G}$ the residual graph has no nodes from V.*

Proof. Assume that a node $i \in V$ remains in the graph after removing $V(\mathcal{K})$. Since the residual graph has no connected node pairs, the corresponding copy of i—i.e., node $\hat{i} \in \hat{V}$—must be part of a star in $\mathcal{K}$, and since $\hat{i}$ is not the hub of any star in $\mathcal{K}$, as $\mathcal{K}_2 = \emptyset$, $\hat{i}$ must be a leaf of a star in $\mathcal{K}$. However, $\hat{i}$ is only adjacent to its copy i in $\hat{G}$, and i is not a hub of a star in $\mathcal{K}$ as it remained in the residual graph, which is clearly a contradiction. □

We will now assume without loss of generality that solution $\mathcal{K}$ contains only stars with hubs from V. Since all nodes from V are either hubs or leaves of the stars in $\mathcal{K}$, the hubs of the stars in $\mathcal{K}$ induce a dominating set of size $b = k$ in G.

To prove sufficiency, assume that graph G has a dominating set N of size k. The following procedure creates a valid solution $\mathcal{K}$ of CSP for CNP over $\hat{G}$. First, select any node i from N. Second, add to $\mathcal{K}$ the star composed of i, and $\mathcal{N}_G(i) \setminus N$. Third, remove node i and the subset of neighbors $\mathcal{N}_G(i) \setminus N$ from G. Fourth, select another node from N and repeat the process until G is empty. Clearly, since $|N| = k$, this procedure takes k iterations. Furthermore, note that the residual graph contains only the nodes from set $\hat{V}$. Since none of these nodes are adjacent, the number of connected node pairs that remains is zero. Dominating set is known to be NP-complete [29], therefore, the proposed reduction proves that CSP is NP-complete as well.

Finally, we show that the decision version of CSP is also NP-complete for the LC connectivity measure. As noted in Theorem 1, the size of the largest component of a graph is at the most one if such a graph is disconnected. Therefore, we can use the same reduction to transform an instance of dominating set into an instance of CSP for LC with $b = k$ and $r = 1$. □

Theorem 3 *CCSP for both the LC and CNP measures is strongly NP-complete.*

Proof of Theorem 3. We begin by proving the result for the LC connectivity measure. The reduction is from Exact Cover; a decision problem that receives as input a collection $\mathcal{C} = \{C_1, \ldots, C_m\}$ of subsets of a set $X = \{x_1, \ldots, x_n\}$ and asks if there is a subcollection $\mathcal{S} \subseteq \mathcal{C}$ such that each element in X is contained in exactly one subset in $\mathcal{S}$. Let $x^k_{(i)}$ be the i^{th} element of subset $C_k \in \mathcal{C}$ (for any ordering). Given an instance of Exact Cover, we construct graph G as follows. First, create a node for each $x_i \in X$. Then, for each element $x^k_{(i)}$ of subset $C_k \in \mathcal{C}$, create a copy node named $\hat{x}^k_{(i)}$, and two dummy nodes $y^k_{(i)}$ and $z^k_{(i)}$. Let $\hat{X}_k$ be the set of all copy nodes for set C_k. Continue by adding edges connecting $x^k_{(i)}$ with $y^k_{(i)}$, $x^k_{(i)}$ with $z^k_{(i)}$, $\hat{x}^k_{(i)}$ with $y^i_{(i)}$, $\hat{x}^k_{(i)}$ with $z^k_{(i)}$, and $y^k_{(i)}$ with $z^k_{(i)}$. Finally, add an arbitrary number of edges connecting the copy nodes in $\hat{X}_k$ so that the resulting subgraph $\hat{G}_k$, induced by the nodes in $\hat{X}_k$, is connected (adding enough edges so that $\hat{G}_k$ becomes a spanning tree for the nodes in $\hat{X}_k$ suffices). Notice that $\hat{G}_k$ is a connected graph with $|C_k|$ nodes. An example of this construction (i.e., gadget) for a given subset C_k is shown in Figure 6.

After this process is repeated for all subsets in $\mathcal{C}$, the resulting graph G has a total of $n + 3\sum_{k=1}^{m} |C_k|$ nodes. Now, let $\mathcal{T}$ comprise all subgraphs $\hat{G}_k$ for $k = 1, \ldots, m$, and all triangles $G[\{\hat{x}^k_{(i)}, y^k_{(i)}, z^k_{(i)}\}]$ and $G[\{x^k_{(i)}, y^k_{(i)}, z^k_{(i)}\}]$ for each $x^k_{(i)} \in C_k$ and all $C_k \in \mathcal{C}$. Set $\mathcal{T}$ contains a total of $m + 2\sum_{k=1}^{m} |C_k|$ subgraphs. Clearly, the number of iterations required to perform the proposed construction is bounded by a polynomial in the sizes of X and $\mathcal{C}$.

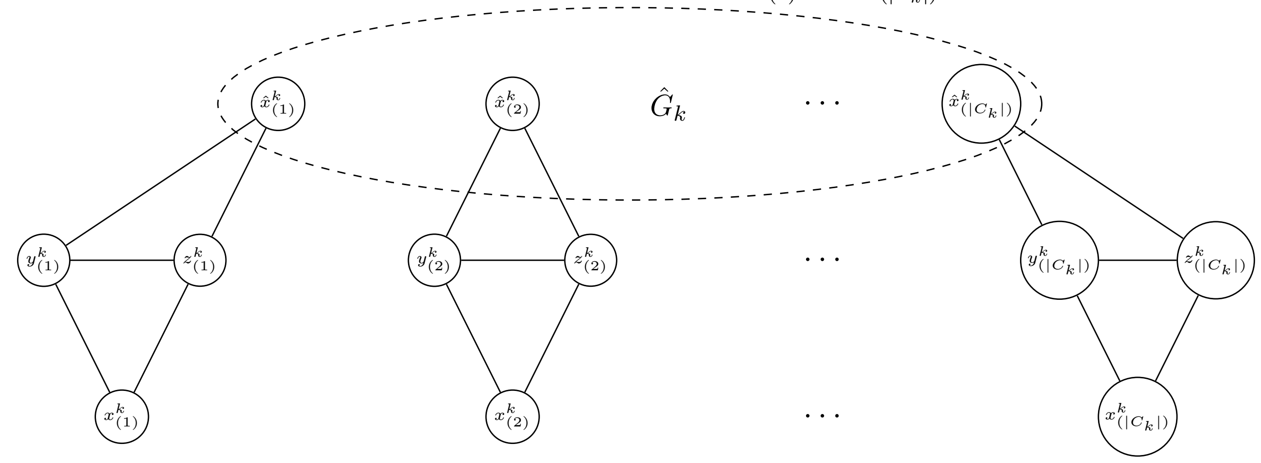
To illustrate the proposed construction, consider the exact cover instance $X = \{x_a, x_b, x_c, x_d, x_e, x_f, x_g\}$; $\mathcal{C} = \{\{x_a, x_b, x_c, x_d\}, \{x_e, x_f, x_g\}, \{x_c, x_d, x_e\}\}$. The resulting graph G is depicted in Figure 7. Notice that $\hat{G}_1 = G[\{\hat{x}^1_a, \hat{x}^1_b, \hat{x}^1_c, \hat{x}^1_d\}]$, $\hat{G}_2 = G[\{\hat{x}^2_e, \hat{x}^2_f, \hat{x}^2_g\}]$, and $\hat{G}_3 = G[\{\hat{x}^3_c, \hat{x}^3_d, \hat{x}^3_e\}]$ can be arbitrarily constructed by adding any set of edges as long as the graphs are connected.

We now prove that there is an exact cover of X with subsets from $\mathcal{C}$ if and only if there is a collection $\mathcal{K}$ of at most $b = \sum_{k=1}^{m}(|C_k| + 1)$ disjoint subgraphs from $\mathcal{T}$ so that the size of the largest component in $\hat{G}[V \setminus V(\mathcal{K})]$ is $r = 0$. To proof necessity, assume that there is such a set $\mathcal{K}$.

Claim 6 *For the gadget associated with a subset $C_k \in \mathcal{C}$, set $\mathcal{K}$ contains either all triangles $G[\{x^k_{(i)}, y^k_{(i)}, z^k_{(i)}\}]$ and no triangle $G[\{\hat{x}^k_{(i)}, y^k_{(i)}, z^k_{(i)}\}]$, or vice versa. That is, all triangles $G[\{\hat{x}^k_{(i)}, y^k_{(i)}, z^k_{(i)}\}]$ and no triangle $G[\{x^k_{(i)}, y^k_{(i)}, z^k_{(i)}\}]$.*

Proof. Assume there is a subset C_k such that, for elements $x^k_{(i)}, x^k_{(j)} \in C_k$, solution $\mathcal{K}$ contains subgraphs $G[\{x^k_{(i)}, y^k_{(i)}, z^k_{(i)}\}]$ and $G[\{\hat{x}^k_{(j)}, y^k_{(j)}, z^k_{(j)}\}]$. Since the size of the largest component of

Figure 6: Gadget for subset $C_k = \{x^k_{(1)}, \ldots, x^k_{(|C_k|)}\}$.



$G[V \setminus V(\mathcal{K})]$ is zero, node $\hat{x}^k_{(i)}$ must belong to another subgraph in $\mathcal{K}$. The only other subgraph in $\mathcal{T}$ that contains $\hat{x}^k_{(i)}$ is $\hat{G}_k$, but $\hat{G}_k$ cannot be in $\mathcal{K}$ because $G[\{x^k_{(i)}, y^k_{(i)}, z^k_{(i)}\}]$ is already in $\mathcal{K}$. This would imply that subgraphs in $\mathcal{K}$ are not disjoint, a contradiction. Also, since $y^k_{(i)}$ and $z^k_{(i)}$ are only contained in subgraphs $G[\{x^k_{(i)}, y^k_{(i)}, z^k_{(i)}\}]$ and $G[\{\hat{x}^k_{(i)}, y^k_{(i)} z^k_{(i)}\}]$, either $\mathcal{K}$ contains all triangles $G[\{x^k_{(i)}, y^k_{(i)}, z^k_{(i)}\}]$ or all triangles $G[\{\hat{x}^k_{(i)}, y^k_{(i)}, z^k_{(i)}\}]$. □

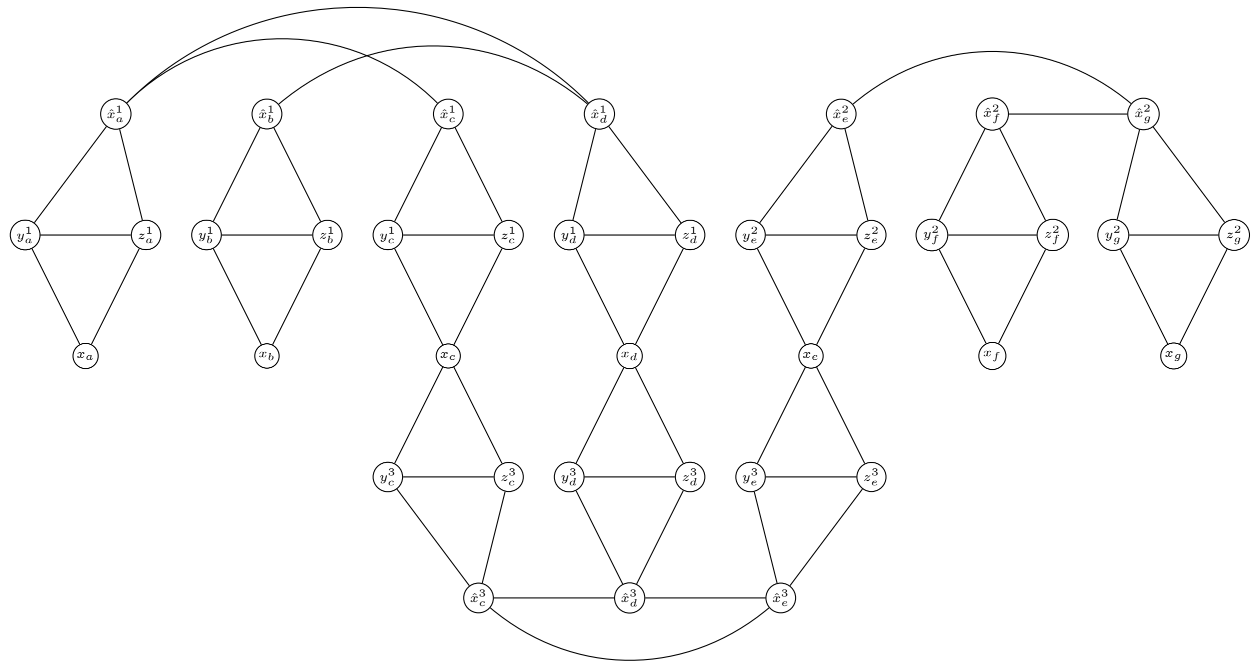
An exact cover $\mathcal{S} \subseteq \mathcal{C}$ from $\mathcal{K}$ can be constructed as follows: add set C_k to $\mathcal{S}$ if $\mathcal{K}$ contains the group of triangles $G[\{x^k_{(i)}, y^k_{(i)}, z^k_{(i)}\}]$. Since all subgraphs in $\mathcal{K}$ are disjoint and the size of the largest component in $G[V \setminus V(\mathcal{K})]$ is zero, the nodes associated with all elements $x_i \in X$ must be covered by exactly one triangle $G[\{x^k_{(i)}, y^k_{(i)}, z^k_{(i)}\}]$. Therefore, the resulting subcollection $\mathcal{S}$ is an exact cover of X.

To prove sufficiency, assume there exists an exact cover given by a subcollection $\mathcal{S} \in \mathcal{C}$. The following procedure creates a valid solution $\mathcal{K}$ of CCSP. For each subset $C_k \in \mathcal{S}$, populate $\mathcal{K}$ by adding graph $\hat{G}_k$ and all triangles $G[\{x^k_{(i)}, y^k_{(i)}, z^k_{(i)}\}]$. Then, for each subset $C_k \notin \mathcal{S}$, expand $\mathcal{K}$ with all triangles $G[\{\hat{x}^k_{(i)}, y^k_{(i)}, z^k_{(i)}\}]$. It is easy to verify that the subgraphs in set $\mathcal{K}$ produce a partition of G. Thus, if the nodes from set $V(\mathcal{K})$ are removed, the resulting graph is empty—i.e., the size of the largest component is zero. Note that, from the gadget associated with each subset $C_k \in \mathcal{C}$, set $\mathcal{K}$ contains either all triangles $G[\{x^k_{(i)}, y^k_{(i)}, z^k_{(i)}\}]$ or all triangles $G[\{\hat{x}^k_{(i)}, y^k_{(i)}, z^k_{(i)}\}]$. Therefore, $\mathcal{K}$ contains in total exactly $\sum_{k=1}^m |C_k|$ such triangles. In addition to that, $\mathcal{K}$ includes the $\hat{G}_k$ graphs for all $C_k \in \mathcal{S}$. Since $|\mathcal{S}| \leq |\mathcal{C}|$, then $|\mathcal{K}| = \sum_{k=1}^m |C_k| + |\mathcal{S}| \leq \sum_{k=1}^m (|C_k| + 1)$. This implies $\mathcal{K}$ is a valid solution to CCSP.

For the CNP case, from any exact cover instance, take the graph G and set $\mathcal{T}$ generated with the procedure described before. Then, add a dummy node and attach it to all the other original nodes in G. Call this new graph G'. Since the new node does not belong to any subgraph in $\mathcal{T}$, the only way to have a residual graph with no connected node pairs is if all the nodes originally in G are removed. Thus, a solution $\mathcal{K} \subseteq \mathcal{T}$ of at most $b = \sum_{k=1}^m (|C_k| + 1)$ disjoint subgraphs of G' exists so that the number of connected node pairs of $G[V \setminus V(\mathcal{K})]$ is $r = 1$, if and only if there is a solution for the exact cover instance. This completes the proof. □

Proposition 1 *Let* $(\bar{\mathbf{z}}, \bar{\mathbf{x}}, \bar{\mathbf{y}})$ *be an optimal solution of the linear relaxation of (11)-(15), in which*

Figure 7: Graph G constructed from the exact cover instance $X = \{x_a, x_b, x_c, x_d, x_e, x_f, x_g\}$; $\mathcal{C} = \{\{x_a, x_b, x_c, x_d\}, \{x_e, x_f, x_g\}, \{x_c, x_d, x_e\}\}$.



(14) is defined by the path constraint set (6)-(7). Then $(\bar{\mathbf{x}}, \bar{\mathbf{y}})$ satisfies the linear relaxation of the formulation with the transitive constraint set (3)-(5).

Proof of Proposition 1. Let i, j, and k be any triplet of nodes in V—all three connected in the original graph G—and $P_1^* \in \arg\min_{P \in \mathcal{P}^{ij}} \{\sum_{l \in V(P)} \bar{x}_l\}$, $P_2^* \in \arg\min_{P \in \mathcal{P}^{jk}} \{\sum_{l \in V(P)} \bar{x}_l\}$ and $P_3^* \in \arg\min_{P \in \mathcal{P}^{ik}} \{\sum_{l \in V(P)} \bar{x}_l\}$. Since constraints (3) are included in (6), we only need to prove that $\bar{x}_i, \bar{x}_j, \bar{x}_k, \bar{y}_{ij}, \bar{y}_{ik}$, and $\bar{y}_{jk}$, satisfy (4). To this end, consider the following two cases. First, suppose that $\sum_{l \in V(P_1^*)} \bar{x}_l \geq 1$, which implies that $\bar{y}_{ij} = 0$ (the same can be assumed for P_2^* or P_3^*). For this case, it suffices to prove that $y_{jk} + y_{ik} \leq 1$, which is equivalent to proving that $\sum_{l \in V(P_2^*)} \bar{x}_l + \sum_{l \in V(P_3^*)} \bar{x}_l \geq 1$, as (6) implies that $\sum_{l \in V(P_2^*)} \bar{x}_l + \bar{y}_{jk} + \sum_{l \in V(P_3^*)} \bar{x}_l + \bar{y}_{ik} \geq 2$. Clearly, if $i \in P_2^*$ or $j \in P_3^*$, $\sum_{l \in V(P_2^*)} \bar{x}_l \geq 1$ or $\sum_{l \in V(P_3^*)} \bar{x}_l \geq 1$, respectively. On the other hand, if $i \notin P_2^*$ and $j \notin P_3^*$, the union of paths P_2^* and P_3^* must then contain a path $P_1' \in \mathcal{P}^{ij}$, which implies that $\sum_{l \in V(P_2^*)} \bar{x}_l + \sum_{l \in V(P_3^*)} \bar{x}_l \geq \sum_{l \in V(P_1')} \bar{x}_l \geq \sum_{l \in V(P_1^*)} \bar{x}_l \geq 1$.

Second, suppose that $\sum_{l \in V(P_1^*)} \bar{x}_l < 1$, $\sum_{l \in V(P_2^*)} \bar{x}_l < 1$, and $\sum_{l \in V(P_3^*)} \bar{x}_l < 1$. Since the objective function minimizes the sum of the $\mathbf{y}$ variables, we have $\sum_{l \in V(P_1^*)} \bar{x}_l + y_{ij} = 1$, $\sum_{l \in V(P_2^*)} \bar{x}_l + y_{jk} = 1$, and $\sum_{l \in V(P_3^*)} \bar{x}_l + y_{ik} = 1$. Furthermore, it is easy to see that $\sum_{l \in V(P_1^*)} \bar{x}_l \leq \sum_{l \in V(P_2^*)} \bar{x}_l + \sum_{l \in V(P_3^*)} \bar{x}_l$, $\sum_{l \in V(P_2^*)} \bar{x}_l \leq \sum_{l \in V(P_1^*)} \bar{x}_l + \sum_{l \in V(P_3^*)} \bar{x}_l$, and $\sum_{l \in V(P_3^*)} \bar{x}_l \leq \sum_{l \in V(P_1^*)} \bar{x}_l + \sum_{l \in V(P_2^*)} \bar{x}_l$, otherwise there exists at least one path in $\mathcal{P}^{ij}, \mathcal{P}^{jk}$, or $\mathcal{P}^{ik}$, respectively, for which the corresponding constraint in (6) is violated. Therefore, replacing $\sum_{l \in V(P_1^*)} \bar{x}_l$, $\sum_{l \in V(P_2^*)} \bar{x}_l$, and $\sum_{l \in V(P_3^*)} \bar{x}_l$ with $1 - y_{ij}$, $1 - y_{jk}$, $1 - y_{ik}$ in the previous expressions yields (4)-(5). $\square$

Proposition 2 *Let $(\bar{\mathbf{z}}, \bar{\mathbf{x}}, \bar{\mathbf{y}})$ be an optimal solution of the linear relaxation of (11)-(15), in which (14) is defined by the path constraint set (6)-(7). Then $(\bar{\mathbf{x}}, \bar{\mathbf{y}})$ satisfies the linear relaxation of the formulation with the neighborhood constraint set (8)-(10).*

Proof of Proposition 2. Similarly as for the transitive constraint set, since constraints (8) are included in (6), we only need to prove that $(\bar{\mathbf{x}}, \bar{\mathbf{y}})$, satisfies (9). Let $i, j \in V$ be any pair of non-adjacent nodes in G. By definition, for any node $k \in \mathcal{N}_G(i)$, $\bar{y}_{jk}$ satisfies the following path constraints $\sum_{l \in V(P)} \bar{x}_l + \bar{y}_{jk} \geq 1$, for all $P \in \mathcal{P}^{jk}$ and, since path $P \cup \{i\} \in \mathcal{P}^{ij}$ for all $P \in \mathcal{P}^{jk}$, $\bar{y}_{ij}$ also satisfies $\sum_{l \in V(P)} \bar{x}_l + \bar{x}_i + \bar{y}_{ij} \geq 1$. Likewise, for any node $q \in \mathcal{N}_G(j)$, $\bar{y}_{iq}$ and $\bar{y}_{ij}$ satisfy the following path constraints $\sum_{l \in V(P)} \bar{x}_l + \bar{y}_{iq} \geq 1$ and $\sum_{l \in V(P)} \bar{x}_l + \bar{x}_j + \bar{y}_{ij} \geq 1$, respectively, for all $P \in \mathcal{P}^{iq}$. Now, notice that since the objective function minimizes the sum of the $\mathbf{y}$ variables, there exists a pair of paths $P_1 \in \mathcal{P}^{jk}$ and $P_2 \in \mathcal{P}^{iq}$ for every $k \in \mathcal{N}_G(i)$ and $q \in \mathcal{N}_G(j)$ for which $\bar{y}_{ij} + \bar{x}_i \geq \bar{y}_{jk} = 1 - \sum_{l \in V(P_1)} \bar{x}_l$ and $\bar{y}_{ij} + \bar{x}_j \geq \bar{y}_{iq} = 1 - \sum_{l \in V(P_2)} \bar{x}_l$, which proves the result. $\square$

Proposition 3 *Separating the path constraints for integer values of $(\bar{\mathbf{x}}, \bar{\mathbf{y}}, \bar{\mathbf{z}})$ takes $O(nm)$ time.*

Proof of Proposition 3. Since $\mathbf{x} \in \{0,1\}^n$, the costs for traversing the nodes are either zero or one. Thus, the solution for each APSP run can be obtained by solving a shortest path problem from every node using Dial's implementation of Dijkstra's algorithm [3]. Solving each shortest path problem using such an implementation takes $O(m)$ time because the largest possible cost is one; hence, the separation takes $O(nm)$ time. $\square$

Proposition 4 *For any clique K of size 3 in G (a triangle), where $V(K) = \{i, j, k\}$, the following inequality is valid. We refer to these inequalities as 3-clique inequalities.*

$$\sum_{\{T \in \mathcal{T} : |V(T) \cap V(K)| = 1\}} z_T + 2 \sum_{\{T \in \mathcal{T} : |V(T) \cap V(K)| > 1\}} z_T + y_{ij} + y_{ik} + y_{jk} \geq 2. \tag{38}$$

Proof of Proposition 4. The validity of these inequalities can be easily derived using the *Chvátal-Gomory (C-G) rounding method* [54]. This is done by simply combining the $\mathcal{T}$-path inequalities for the paths induced by the edges of the clique, i.e., the inequalities for the paths $i - j, j - k$, and $i - k$, which are given below.

$$\sum_{\{T \in \mathcal{T} : V(T) \cap V(K) = \{i\}\}} z_T + \sum_{\{T \in \mathcal{T} : V(T) \cap V(K) = \{j\}\}} z_T + \sum_{\{T \in \mathcal{T} : |V(T) \cap V(K)| > 1\}} z_T + y_{ij} \geq 1, \tag{47}$$

$$\sum_{\{T \in \mathcal{T} : V(T) \cap V(K) = \{i\}\}} z_T + \sum_{\{T \in \mathcal{T} : V(T) \cap V(K) = \{k\}\}} z_T + \sum_{\{T \in \mathcal{T} : |V(T) \cap V(K)| > 1\}} z_T + y_{ik} \geq 1, \tag{48}$$

$$\sum_{\{T \in \mathcal{T} : V(T) \cap V(K) = \{j\}\}} z_T + \sum_{\{T \in \mathcal{T} : V(T) \cap V(K) = \{k\}\}} z_T + \sum_{\{T \in \mathcal{T} : |V(T) \cap V(K)| > 1\}} z_T + y_{jk} \geq 1. \tag{49}$$

The combination of these three inequalities, using $1/2$ as the multiplier for all of them is then rounded up to obtain the desired inequality. $\square$

Proposition 5 *For any chordless cycle C of size 4 in G, where $V(C) = \{i, j, k, l\}$ and $E(C) = \{\{i, j\}, \{j, k\}, \{k, l\}, \{i, l\}\}$, the following inequality is valid. We refer to these inequalities as 4-cycle inequalities.*

$$\sum_{\{T \in \mathcal{T} : |V(T) \cap V(C)| = 1\}} z_T + 2 \sum_{\{T \in \mathcal{T} : |V(T) \cap V(C)| > 1\}} z_T + y_{ik} + y_{jl} \geq 2. \tag{39}$$

Proof of Proposition 5. We use the C-G rounding method for this proof as well. The $\mathcal{T}$-path inequalities for the two paths connecting nodes i and k (i.e., the paths $i-j-k$ and $i-l-k$) and the $\mathcal{T}$-path inequalities connecting nodes j and l (i.e., the paths $j-k-l$ and $j-i-l$), which are all given below, can be combined to produced the proposed inequality.

$$\sum_{\{T\in\mathcal{T}:V(T)\cap V(C)=\{i\}\}} z_T + \sum_{\{T\in\mathcal{T}:V(T)\cap V(C)=\{j\}\}} z_T + \sum_{\{T\in\mathcal{T}:V(T)\cap V(C)=\{k\}\}} z_T + \sum_{\{T\in\mathcal{T}:V(T)\cap V(C)>1\}} z_T + y_{ik} \geq 1, \tag{50}$$

$$\sum_{\{T\in\mathcal{T}:V(T)\cap V(C)=\{i\}\}} z_T + \sum_{\{T\in\mathcal{T}:V(T)\cap V(C)=\{l\}\}} z_T + \sum_{\{T\in\mathcal{T}:V(T)\cap V(C)=\{k\}\}} z_T + \sum_{\{T\in\mathcal{T}:V(T)\cap V(C)>1\}} z_T + y_{ik} \geq 1, \tag{51}$$

for paths $i-j-k$ and $i-l-k$, and

$$\sum_{\{T\in\mathcal{T}:V(T)\cap V(C)=\{j\}\}} z_T + \sum_{\{T\in\mathcal{T}:V(T)\cap V(C)=\{k\}\}} z_T + \sum_{\{T\in\mathcal{T}:V(T)\cap V(C)=\{l\}\}} z_T + \sum_{\{T\in\mathcal{T}:V(T)\cap V(C)>1\}} z_T + y_{jl} \geq 1, \tag{52}$$

$$\sum_{\{T\in\mathcal{T}:V(T)\cap V(C)=\{j\}\}} z_T + \sum_{\{T\in\mathcal{T}:V(T)\cap V(C)=\{i\}\}} z_T + \sum_{\{T\in\mathcal{T}:V(T)\cap V(C)=\{l\}\}} z_T + \sum_{\{T\in\mathcal{T}:V(T)\cap V(C)>1\}} z_T + y_{jl} \geq 1, \tag{53}$$

for paths $j-k-l$ and $j-i-l$. The combination of these four inequalities, using 1/3 as the multiplier for all of them is rounded up to obtain the desired inequality. $\square$

B Branching Rule and Stabilization Techniques

B.1 Branching rule

One of the major difficulties that arise when solving a problem via branch-and-price is defining the branching rule. There are many reasons why the standard 0/1 branching (i.e., fixing a fractional variable to either zero or one) is undesirable in this context. First of all, when fixing a variable z^T to one, for any given node structure $T \in \mathcal{T}'$, the corresponding variables x_i for the nodes that comprise T (i.e., $i \in V(T)$) must also be fixed to one. Since those nodes cannot be covered again by any other node structure—as for constraints (12)—all the additional node structures in $\mathcal{T}$ containing at least one of those nodes must then be discarded in the current branch (i.e., the corresponding variables are fixed to zero). Hence, the number of variables to be considered in that branch is significantly reduced. Conversely, when z^T is fixed to zero, only one of potentially many variables is fixed. This particular behavior results in a highly unbalanced, and generally inefficient, branching tree. Furthermore, when fixing a variable to zero, we must ensure that the pricing subproblem does not produce the same variable again. This is generally done by either finding the next best solution (possibly the kth best after k branches), or by including additional constraints in the subproblem that are often difficult to handle by specialized algorithms. In any of both cases, solving the pricing problem becomes remarkably harder.

An alternative option is using a variation of the so-called Ryan-Foster branching rule [7, 60, 11], which has been widely popular in the context of branch-and-price for solving reformulations of many well-known combinatorial optimization problems as set partitioning problems (SPPs)—e.g., vehicle routing problems [27] and multidimensional assignment problems [76], among others. The SPP is defined by a set of elements U and a collection $\mathcal{S}$ of subsets of U, and it asks for a subcollection $\mathcal{S}' \subseteq \mathcal{S}$ so that the subsets in $\mathcal{S}'$ induce a partition of U and a given cost function $c\colon \mathcal{S} \to \mathbb{R}$ evaluated on $\mathcal{S}'$ is minimized. In the standard SPP formulation, there is a binary variable for each subset $S \in \mathcal{S}$ that takes the value of one if set S belongs to $\mathcal{S}'$ and zero, otherwise. Since each element $i \in U$ must be contained in exactly one of the subsets in $\mathcal{S}'$, the sum of the variables associated with the subsets that contain i must be equal to one. Given a fractional solution of the SPP formulation, there must exist an element $i \in U$ that is partially covered by at least two different sets, say S_1 and S_2 (i.e., the corresponding variables for S_1 and S_2 are fractional). Furthermore, since S_1 and S_2 are different, there exists an element $j \in U, j \neq i$ that is either contained in S_1 and not in S_2, or vice versa. The Ryan-Foster rule then works by generating two branches, one that restricts both elements i and j to be in the same subset and one that force them to be in different subsets. It is easy to see that the given fractional solution is no longer valid, while any feasible integer solution belongs to one of the two branches.

The branching rule we propose inherits from Ryan-Foster the idea of enforcing two fractionally covered nodes to be part of the same or different critical node structures; but, it aslo considers the fact that not all the nodes are necessarily covered by the optimal set of critical node structures (i.e., not all the nodes are removed from G). Before describing the proposed branching rule, we state the following proposition, which provides an insight regarding the variables that must be considered for branching.

Proposition 7 *In formulations (11)-(15) and (16)-(18), when the path constraint set (6)-(7) is used to describe Ω, the integrality constraints of the $\mathbf{y}$ variables can be relaxed and the resulting optimal solution is either integral or it can be trivially converted into an integral solution with the same objective value.*

Proof. First, consider an optimal solution $(\mathbf{x}, \mathbf{y}, \mathbf{z})$ of formulation (11)-(15) obtained after dropping the integrality constraints for the $\mathbf{y}$ variables. For this case, one only need to notice that since the objective function minimizes the sum of the $\mathbf{y}$ variables, constraints (6) guarantee that variable y_{ij} for $\{i,j\} \in \binom{V}{2}$ will take the value of zero, unless there is at least one path $P \in \mathcal{P}^{ij}$ connecting i and j that remains in the graph after the critical node structures are removed, a case in which $y_{ij} = 1$ (the minimum possible value y_{ij} can take to satisfy the corresponding path constraint (6)).

Second, consider an optimal solution $(\mathbf{x}, \mathbf{y}, \mathbf{z}, u)$ of formulation (16)-(18) obtained after dropping the integrality constraints for the $\mathbf{y}$ variables. For this case, because the objective function minimizes the size of the largest remaining component, unless a node i belongs such a component (in such a case all the $y_{ij} \in [0,1]$ for all $j \in V$, for the same reason described above), it is possible for some y_{ij} to be fractional in an optimal solution. Such a fractional variable can then be replaced by $y_{ij} = \min\{0, \min_{P\in\mathcal{P}^{ij}}\{1 - \sum_{k\in V(P)} x_k,\}\}$. The resulting solution is also optimal and integral. □

Proposition 7 implies that only the variables that state if a given node structure is removed from the graph (i.e., the $\mathbf{z}$ variables) and, consequently, the variables that describe if a node is removed as part of one of such node structures (i.e., the $\mathbf{x}$ variables) are required to be considered for branching. The proposed branching is general enough to be usable independently of the type of critical node structures in $\mathcal{T}$ (e.g., cliques, stars, connected node subsets, etc). In fact, if $\mathcal{T} = V$, the branching rule reduces to the traditional 0/1 branching over the $\mathbf{x}$ variables.

Given a solution of MP_{CNP} or MP_{LC} with fractional values for $\mathbf{x}$ and $\mathbf{z}$, we consider two cases as possible branching options:

Case 1: Following the main structure of Ryan-Foster, we consider as candidates for branching any pair of nodes i and j of a fractionally removed critical node structure $T \in \mathcal{T}'$ (i.e., a node structure T for which $a_i^T = a_j^T = 1$ and $z^T \in (0,1)$). We create one branch, referred to as `SAME`, in which we force i and j to be removed from G as part of the same critical node structure. In this branch we fix $x_i = x_j = 1$ and $z_T = 0$, for each node structure $T \in \mathcal{T}$ that contains either i or j, but not both (i.e., any $T \in \mathcal{T}$ for which $a_i^T + a_j^T = 1$). We then create a second branch, named `DIFF`, in which we force i and j to not be present in the same critical node structure by fixing $z_T = 0$, for each $T \in \mathcal{T}$ that contains both i and j (i.e., any $T \in \mathcal{T}$ for which $a_i^T + a_j^T = 2$).

Once we generate the new branches, we also propagate the respective branching requirements in the pricing subproblem to guarantee that any node structure $T \in \mathcal{T} \setminus \mathcal{T}'$ generated thereafter satisfies the corresponding branching restrictions. The way in which such restrictions are propagated depends significantly on the technique used for solving the pricing subproblem. For instance, if the pricing subproblem is solved via mathematical programming, constraints that enforce $a_i^T = a_j^T$ (for `SAME`) and $a_i^T + a_j^T \leq 1$ (for `DIFF`) must be added for each branch and node pairs on the corresponding subproblem formulation. In general, any specialized algorithm used to price new variables must enforce such constraints (see Section 7) as well.

Case 2: The second case is only applicable if the set of node structures $\mathcal{T}$ contains at least one member T that is defined by a single node $i \in V$ (i.e., $V(T) = \{i\}$). For example, any clique of size one, if $\mathcal{T}$ comprises all cliques in G. Suppose then that $T \in \mathcal{T}'$ is one of those structures with $V(T) = \{i\}$, $z_T \in (0, 1)$, and $x_i = z_T$. In other words, node i is itself a critical node structure that is partially removed from the graph. For this case, we create one branch in which we force node i to remain in the graph by fixing $x_i = 0$ and, consequently, $z_T = 0$ for each node structure $T \in \mathcal{T}'$ that contains i (i.e., any $T \in \mathcal{T}'$ for which $a_i^T = 1$). As for Case 1, this requirement must be propagated in the subsequent pricing rounds of this branch, ensuring that $a_i^T = 0$ for any new T generated thereafter. A second branch is created to force node i to be removed from G as part of a critical node structure by fixing $x_i = 1$, which combined with constraint (13), ensures that a node structure that contains node i is removed (i.e, $\sum_{T \in \mathcal{T}'} a_i^T z_T = 1$).

Among all the candidate fractional variables that satisfy the above conditions, we select the branching variables using the procedure described in Algorithm 2. The proposed variable selection can be seen as the analogous version of selecting the fractional variable whose value is closer to 0.5 in the traditional 0/1 branching. We select for branching either the pair of nodes $\{i, j\}$, whose value of $\sum_{\{T \in \mathcal{T}' : \{i,j\} \in \binom{V(T)}{2}\}} z_T$ is the closest to 0.5 (Case 1) or, when single-node node structures exist in $\mathcal{T}'$, the node i whose z_T is closest to 0.5, given that $V(T) = \{i\}$ (Case 2).

Algorithm 2 A procedure that selects the nodes for which the corresponding variables $\mathbf{x}$ and $\mathbf{z}$ will be branched on.

Input: Graph $G = (V, E)$, the set of node structures $\mathcal{T}'$, and the current value of variables $\mathbf{z}$. **Output:** A pair of nodes $\{i, j\}$ or a node i selected for branching.

```
1: $\omega_{ii} \leftarrow 0, \forall i \in V$
2: $\omega_{ij} \leftarrow 0, \forall \{i, j\} \in \binom{V}{2}$
3: for all $T \in \mathcal{T}'$ such that $z_T \in (0, 1)$ do
4:     if $|V(T)| = 1$, and $i \in V(T)$ has not been selected for branching on any of the parent branches then
5:         $\omega_{ii} \leftarrow z_T$
6:     else
7:         for all $\{i, j\} \in \binom{V(T)}{2}$ so that pair $\{i, j\}$ has not been selected for branching on any of the parent branches
           do
8:             $\omega_{ij} \leftarrow \omega_{ij} + z_T$
9:         end for
10:    end if
11: end for
12: $\hat{i} \leftarrow \arg\min_{i \in V} \{\max\{\omega_{ii}, 1 - \omega_{ii}\}\}$
13: $\{\hat{i}, \hat{j}\} \leftarrow \arg\min_{\{i,j\} \in \binom{V}{2}} \{\max\{\omega_{ij}, 1 - \omega_{ij}\}\}$
14: if $\max\{\omega_{\hat{i}\hat{i}}, 1 - \omega_{\hat{i}\hat{i}}\} \leq \max\{\omega_{\hat{i}\hat{j}}, 1 - \omega_{\hat{i}\hat{j}}\}$ then
15:     return $\hat{i}$
16: else
17:     return $\{\hat{i}, i\hat{j}\}$
18: end if
```

B.2 Stabilization methods

Solving the linear relaxations of formulations (11)-(15) and (16)-(18) via column generation could be a rather slow process unless a proper stabilization procedure is applied. In addition to the well-known tailing-off effect, prevalent in most column generation applications [45], the degeneracy of these formulations may cause a major delay in the process of declaring optimality. One of the principal causes of this behavior is the instability of the dual variables, for those tend to oscillate without smoothly converging to their respective optimal levels [45]. This behavior is particularly emphasized in early iterations when fewer dual constraints have been generated and thus little information is known about the dual problem. Since the quality of the newly generated columns depends heavily on the dual values and the fact that the pricing subproblems can be rather difficult to solve (see Section 7), the strong effects on the expected running time are of enough relevance for considering the use of a stabilization mechanism in an attempt to reduce this erratic behavior.

Several methods have been proposed to overcome this issue [2, 48, 24, 59]. The main purpose of these techniques is to scale down the strong fluctuations of the dual variables throughout the column generation stage. More specifically, the so-called box stabilization methods [2, 48] attempt to limit strong changes of the dual variables by either (1) imposing dynamic bounds on them based on the values of those in the previous iterations, or by (2) introducing additional costs in the MPs trying to minimize the norm of the difference between dual values between iterations. Alternatively, interior-point methods like the one in [59] try to generate dual values corresponding to solutions in the interior of the dual polyhedron. Because of degeneracy, there exist an infinite number of such optimal interior solutions that can be selected for the pricing procedure. In general, using interior values of the dual variables is known to produce a balanced behavior in column generation routines [12, 53], thus leading to a better performance.

In this paper we use an approach similar to the one proposed in [59]. The idea is rather simple to implement, with the minor caveat of having to explicitly maintain a modified version of the DP in memory during the early stages of the branching tree. The procedure proposed in [59] generates several alternative dual optimal extreme points by sequentially solving a modified version of the dual problem (i.e., a variation that guarantees that the resulting solutions are dual feasible and are complementary to the primal solution), using as objective function a weighted sum of the dual variables with random multipliers. These extreme dual solutions are then used in a convex combination to produce the final set of values that is later passed to the pricing subproblem. The main drawback of this approach when used in a branch-price-and-cut framework is that DP can become quite large as both new rows (associated with the new priced variables in MP) and columns (associated with the separated path constraints in MP) are constantly being generated. Therefore, the time required for solving a large DP several times per iteration can potentially outgrow the benefits of the stabilization procedure.

To overcome this issue, we reduce the number of times the variation of DP is solved to one. Let $(\bar{\alpha}, \bar{\beta}, \bar{\gamma})$ be the dual solution obtained from MP. The idea is to produce a dual solution $(\hat{\alpha}, \hat{\beta}, \hat{\gamma})$ sufficiently different from $(\bar{\alpha}, \bar{\beta}, \bar{\gamma})$, so that the resulting convex combination between both solutions produces a somehow good interior point. Ideally, one would like to find a solution that maximizes a given norm between both solutions (e.g., $\|(\bar{\alpha}, \bar{\beta}, \bar{\gamma}) - (\hat{\alpha}, \hat{\beta}, \hat{\gamma})\|_1$), which in turn could be rather difficult [62]. Instead, we solve a variation of DP in which we maximize the sum of the non-basic variables of DP—including the slack variables—whose reduced cost is zero. That information is easily obtainable after solving MP. We acknowledge that producing more extreme points can potentially yield a more central point, but in turn may require a larger computational effort. We limit our discussion to the CNP connectivity measure.

Let s^z_T for all $T \in \mathcal{T}'$, s^x_i for all $i \in V$, and s^y_{ij} for all $\{i, j\} \in \binom{V}{2}$ be the slack variables of the constraints in DP associated with the primal variables $\mathbf{z}, \mathbf{x}$, and $\mathbf{y}$, respectively. Let I_α be a binary parameter that indicates with one whether variable α is non-basic and has a reduced cost of zero in the optimal basis obtained from MP. Furthermore, let $\mathcal{P}_\gamma^{ij}$ for all $\{i,j\} \in \binom{V}{2}$, $\mathcal{T}'_z$, V_x, and V_y be the indices of dual variables and slacks, γ, $\mathbf{s}^z$, $\mathbf{s}^x$, and $\mathbf{s}^y$, respectively, that are non-basic and have reduced cost equal to zero in the current optimal basis obtained from MP. Then, the proposed formulation follows:

$$\max \quad I_\alpha \alpha + \sum_{\{i,j\}\in\binom{V}{2}} \sum_{P\in\mathcal{P}_\gamma^{ij}} \gamma_{ij}^P + \sum_{T\in\mathcal{T}'_z} s_T^z + \sum_{i\in V_x} s_i^x + \sum_{\{i,j\}\in V_y} s_{ij}^y \tag{54}$$

$$\text{s.t.} \quad -b\alpha + \sum_{\{i,j\}\in\binom{V}{2}} \sum_{P\in\mathcal{P}^{ij}} \gamma_{ij}^P = \sum_{\{i,j\}\in\binom{V}{2}} y_{ij}^*; \tag{55}$$

$$-c(T)\alpha + \sum_{i\in V} a_i^T \beta_i + s_T^z = 0,\ T \in \mathcal{T}'; \tag{56}$$

$$-\beta_i + \sum_{j,k\in V} \sum_{\{P\in\mathcal{P}^{jk} | i\in V(P)\}} \gamma_{jk}^P + s_i^x = 0,\ i \in V; \tag{57}$$

$$\sum_{P\in\mathcal{P}^{ij}} \gamma_{ij}^P + s_{ij}^y = 1,\ \{i,j\} \in \binom{V}{2}; \tag{58}$$

$$\alpha \geq 0; \tag{59}$$

$$\gamma_{ij}^P \geq 0,\ \{i,j\} \in \binom{V}{2},\ P \in \mathcal{P}^{ij}; \tag{60}$$

$$s_T^z \geq 0, T \in \mathcal{T}'; \tag{61}$$

$$s_i^x \geq 0, i \in V; \tag{62}$$

$$s_{ij}^y \geq 0, \{i,j\} \in \binom{V}{2}. \tag{63}$$

Here, in constraint (55) we equate the objective of DP to $\sum_{\{i,j\}\in\binom{V}{2}} y_{ij}^*$, which corresponds to the optimal objective of MP. By strong duality, this guarantees that the resulting solution is optimal as well. The rest are the other dual constraints after adding the corresponding slack variables. It is easy to see that formulation (54)-(63) is always bounded, hence it will always provide an optimal dual solution.

C Computational Results (Complete Tables)

Table 6: Computational times, upper and lower bounds, and optimality gaps for detecting critical cliques over the randomly generated graphs while minimizing the CNP objective.

			`URG`				`BA`				`WS`			
n	m	b	Time (s)	UB	LB	Gap %	Time (s)	UB	LB	Gap %	Time (s)	UB	LB	Gap %
		1	1	334	334	0	1	220	220	0	0	342	342	0
	60	3	6	95	95	0	1	21	21	0	0	143	143	0
		5	2	14	14	0	0	3	3	0	0	20	20	0
		1	0	325	325	0	0	246	246	0	0	317	317	0
30	90	3	9	153	153	0	7	21	21	0	0	144	144	0
		5	4	24	24	0	8	1	1	0	0	18	18	0
		1	0	276	276	0	0	177	177	0	0	276	276	0
	180	3	36	100	100	0	29	28	28	0	4	91	91	0
		5	16	7	7	0	66	1	1	0	0	1	1	0
		1	2	584	584	0	1	346	346	0	0	654	654	0
	80	3	12	360	360	0	21	81	81	0	1	326	326	0
		5	56	98	98	0	31	6	6	0	0	120	120	0
		1	2	630	630	0	1	466	466	0	0	630	630	0
40	120	3	17	425	425	0	60	151	151	0	8	345	345	0
		5	309	234	234	0	863	24	24	0	3	154	154	0
		1	1	595	595	0	1	476	476	0	0	572	572	0
	240	3	127	342	342	0	131	128	128	0	94	276	276	0
		5	3,209	159	108	49	7,200	13	13	0	37	56	56	0
		3	394	1,086	1,086	0	272	264	264	0	3	1,091	1,091	0
	120	5	3,785	606	606	0	1,685	85	85	0	7	597	597	0
		10	3,895	32	32	0	216	4	4	0	1	59	53	13
		3	512	1,192	1,192	0	1,139	593	593	0	21	1,113	1,113	0
60	180	5	5,286	836	772	9	6,324	237	237	0	203	667	628	6
		10	7,200	74	52	43	5,685	6	6	0	3	63	30	105
		3	1,235	1,112	1,112	0	2,287	691	691	0	1,048	975	975	0
	360	5	7,200	754	600	26	7,200	269	146	98	7,200	529	315	68
		10	7,200	160	12	1,323	7,200	3	3	0	21	31	1	4,011
		3	638	1,673	1,673	0	1,908	639	639	0	25	1,565	1,565	0
	140	5	6,328	1,153	1,011	15	6,303	223	223	0	35	1,044	976	7
		10	7,200	108	90	20	4,094	10	10	0	2	111	81	38
		3	951	1,711	1,711	0	1,449	1,130	1,130	0	245	1,598	1,598	0
70	210	5	6,717	1,309	1,125	16	7,200	471	471	0	1,582	1,082	1,028	5
		10	7,200	421	100	324	7,200	14	14	0	4	243	58	315
		3	2,418	1,653	1,653	0	4,121	1,052	1,052	0	539	1,396	1,396	0
	420	5	7,200	1,225	1,000	23	7,200	591	295	99	7,200	903	655	38
		10	7,200	486	53	822	7,200	13	10	44	4,835	101	8	1,240
		5	7,200	2,936	2,427	21	7,200	1,017	1,017	0	3,060	2,774	2,204	26
	200	10	7,200	1,283	329	283	7,200	57	57	0	484	793	294	172
		15	7,200	109	65	65	7,200	13	13	0	42	134	69	93
		5	7,200	3,376	2,897	17	7,200	1,702	1,546	11	7,084	2,986	2,368	26
100	300	10	7,200	2,396	679	253	7,200	114	114	0	2,192	1,278	256	401
		15	7,200	703	101	590	7,200	22	20	11	3,231	212	42	406
		5	7,200	3,269	2,785	17	7,200	2,302	1,598	45	7,200	2,628	1,971	33
	600	10	7,200	2,324	626	272	7,200	935	134	578	7,200	1,259	122	940
		15	7,200	1,295	59	2,099	7,200	16	11	56	6,869	163	2	9,190

Table 7: Computational times, upper and lower bounds, and optimality gaps for detecting critical cliques over the randomly generated graphs while minimizing the LC objective.

			URG				BA				WS			
n	m	b	Time (s)	UB	LB	Gap %	Time (s)	UB	LB	Gap %	Time (s)	UB	LB	Gap %
		1	1	26	26	0	1	21	21	0	2	24	24	0
	60	3	9	13	13	0	1	5	5	0	555	14	14	0
		5	12	4	4	0	1	2	2	0	58	2	2	0
		1	1	26	26	0	1	23	23	0	1	27	27	0
30	90	3	60	18	18	0	3	5	5	0	26	16	16	0
		5	97	6	6	0	0	1	1	0	14	4	4	0
		1	2	24	24	0	1	19	19	0	1	26	26	0
	180	3	422	15	15	0	50	6	6	0	65	17	17	0
		5	2,503	3	2	52	4	2	2	0	84	5	5	0
		1	4	35	35	0	1	26	26	0	7	36	36	0
	80	3	227	27	27	0	6	10	10	0	317	26	26	0
		5	161	11	11	0	1	2	2	0	3,300	16	16	0
		1	5	36	36	0	3	31	31	0	8	34	34	0
40	120	3	660	30	30	0	64	17	17	0	5,904	24	22	11
		5	7,200	22	15	48	29	4	4	0	7,202	11	3	267
		1	7	35	35	0	8	31	31	0	6	37	37	0
	240	3	5,445	27	26	4	1,720	16	16	0	116	25	25	0
		5	7,200	18	6	196	3,801	4	3	40	485	13	13	0
		3	6,065	47	45	6	39	20	20	0	6,682	47	45	5
	120	5	7,200	35	22	61	279	9	9	0	5,700	32	23	42
		10	6,990	4	3	67	10	2	2	0	7,200	6	2	142
		3	7,200	50	42	18	1,772	35	35	0	7,200	48	38	26
60	180	5	7,200	41	24	74	2,969	18	16	15	7,200	37	21	76
		10	7,200	8	2	264	19	2	2	0	7,200	8	2	322
		3	7,200	48	38	25	7,200	38	27	39	7,200	45	33	35
	360	5	7,200	40	21	93	7,200	23	7	224	7,200	33	13	166
		10	7,202	19	1	1500	2,882	2	2	0	7,200	7	0	1,502
		3	7,200	58	49	19	988	34	34	0	7,200	57	48	19
	140	5	7,200	49	27	82	1,524	17	17	0	7,200	46	25	86
		10	7,200	8	3	142	543	2	2	0	7,200	8	3	149
		3	7,200	59	52	14	7,200	48	39	25	7,200	57	46	25
70	210	5	7,200	52	32	65	5,521	28	19	45	7,200	47	25	87
		10	7,202	28	4	600	4,096	3	3	26	7,200	17	3	572
		3	7,200	58	49	18	7,200	46	35	31	7,200	53	39	35
	420	5	7,200	50	30	69	7,200	35	14	159	7,200	44	20	119
		10	7,200	32	3	1144	7,200	4	1	212	7,200	13	1	1,657
		5	7,200	79	48	67	5,305	42	29	49	7,200	79	43	83
	200	10	7,200	44	9	388	4,813	5	3	58	7,200	30	7	304
		15	7,200	9	2	293	6,234	3	1	96	7,200	10	2	347
		5	7,200	84	59	43	7,200	61	34	79	7,200	78	48	62
100	300	10	7,200	66	16	302	7,200	8	4	133	7,200	46	7	574
		15	7,200	39	3	1260	7,200	3	1	117	7,200	17	2	896
		5	7,200	82	56	47	7,200	69	38	86	7,200	73	39	89
	600	10	7,200	64	13	404	7,200	38	5	729	7,200	49	4	1,249
		15	7,200	51	2	2284	7,200	5	1	337	7,200	16	0	3,455

Table 8: Number of calls within the branch-price-and-cut and overall execution times of path constraint separation algorithm as and the pricing subproblem for the CNP objective.

			Path Constraint Separation						Pricing Subproblem					
			`URG`		`BA`		`WS`		`URG`		`BA`		`WS`	
n	m	b	Calls	Time	Calls	Time	Calls	Time	Calls	Time	Calls	Time	Calls	Time
		1	136	0	61	0	160	0	173	0	72	0	206	0
	60	3	446	0	72	0	1,607	0	586	0	94	0	2,148	0
		5	420	0	45	0	2,533	1	578	0	60	0	3,393	0
		1	68	0	54	0	79	0	88	0	76	0	108	0
30	90	3	1,618	0	128	0	2,182	0	2,125	0	178	0	2,883	0
		5	4,801	1	11	0	4,132	1	6,319	0	20	0	5,393	0
		1	13	0	24	0	30	0	49	0	53	0	93	0
	180	3	4,860	1	1,812	0	6,761	1	6,579	0	2,847	0	9,142	0
		5	30,828	6	89	0	5,025	1	46,713	3	235	0	10,116	1
		1	55	0	61	0	230	0	61	0	66	0	298	0
	80	3	2,658	1	270	0	1,392	1	3,572	0	354	0	1,834	0
		5	8,242	4	113	0	12,597	6	10,853	0	153	0	16,561	1
		1	67	0	66	0	142	0	81	0	90	0	192	0
40	120	3	4,607	2	1,695	1	1,450	1	6,497	0	2,272	0	1,874	0
		5	66,338	32	1,703	1	39,391	19	89,973	6	2,218	0	52,097	3
		1	34	0	28	0	40	0	96	0	71	0	116	0
	240	3	6,773	3	16,123	7	9,867	5	9,933	1	22,531	2	13,319	1
		5	521,443	235	13,056	5	394,449	173	674,111	152	18,381	1	508,704	88
		3	6,545	11	335	0	9,861	17	8,810	1	441	0	13,149	1
	120	5	44,658	71	1,186	2	111,358	177	59,478	5	1,566	0	148,685	14
		10	66,770	87	181	0	912,459	1,182	90,202	8	315	0	1,212,766	461
		3	23,465	40	716	1	11,682	19	34,468	3	964	0	15,586	1
60	180	5	140,067	221	9,739	15	116,211	179	185,366	21	12,927	1	153,983	15
		10	676,650	929	1,154	1	1,082,407	1,452	881,725	238	1,696	0	1,362,241	550
		3	29,156	45	25,156	38	19,269	29	45,302	5	35,110	4	26,535	3
	360	5	168,989	252	262,045	382	153,087	221	225,376	29	323,123	97	187,168	27
		10	357,271	512	3,289	4	824,580	1,111	422,623	125	6,646	1	991,801	1,093
		3	31,970	83	1,010	3	7,963	20	44,052	5	1,340	0	10,614	1
	140	5	105,965	261	2,770	6	111,112	288	138,868	17	3,679	0	148,007	17
		10	491,889	1,045	566	1	927,396	1,897	642,089	168	848	0	1,201,375	482
		3	12,281	32	6,220	15	11,141	29	17,973	2	8,371	1	14,825	2
70	210	5	99,414	249	34,276	83	86,958	211	131,162	17	45,819	4	114,194	13
		10	165,137	394	986	2	335,234	739	198,162	34	1,481	0	415,051	98
		3	29,732	74	8,226	20	22,850	55	46,105	6	11,692	1	31,435	4
	420	5	87,438	212	132,509	308	110,682	254	118,966	16	163,395	31	137,443	20
		10	116,341	263	634,859	1,293	375,535	793	135,394	23	862,682	761	442,129	147
		5	30,925	220	36,474	252	23,840	271	39,788	8	48,464	8	30,894	8
	200	10	73,113	515	56,626	323	84,868	544	87,838	20	75,330	12	102,307	24
		15	641,840	3,744	5,662	32	638,130	3,649	823,320	279	8,943	1	818,196	239
		5	23,080	165	38,566	269	29,285	204	29,684	7	51,220	9	37,673	8
100	300	10	14,772	106	144,586	834	47,498	308	17,701	3	190,568	31	56,909	12
		15	90,446	579	385,167	2,174	334,054	1,963	108,027	24	536,125	161	410,229	147
		5	15,343	109	41,987	283	34,315	231	20,484	5	53,469	14	45,144	11
	600	10	15,679	104	86,346	539	39,016	240	19,294	4	103,985	29	46,351	11
		15	39,285	246	486,936	2,799	245,187	1,475	45,428	10	664,238	588	297,161	110

Table 9: Number of calls within the branch-price-and-cut and overall execution times of path constraint separation algorithm as and the pricing subproblem for the LC objective.

			Path Constraint Separation						Pricing Subproblem					
			`URG`		`BA`		`WS`		`URG`		`BA`		`WS`	
n	m	b	Calls	Time	Calls	Time	Calls	Time	Calls	Time	Calls	Time	Calls	Time
		1	86	0	140	0	63	0	84	0	63	0	216	0
	60	3	1,463	1	2,098	0	482	0	649	0	63,170	26	101,173	16
		5	6,614	3	8,898	0	349	0	489	0	9,401	3	17,885	2
		1	79	0	138	0	59	0	97	0	85	0	140	0
30	90	3	7,532	3	12,028	0	823	0	1,097	0	3,837	2	5,765	1
		5	29,842	13	39,511	2	35	0	72	0	7,509	3	10,084	1
		1	56	0	193	0	33	0	122	0	73	0	135	0
	180	3	44,494	18	72,501	3	9,581	4	13,305	1	8,984	4	14,221	2
		5	505,614	211	674,954	868	1,229	0	2,690	0	33,412	14	44,352	7
		1	88	0	137	0	53	0	65	0	120	0	215	0
	80	3	10,150	10	16,049	1	786	1	1,065	0	11,984	12	19,663	3
		5	22,018	22	28,879	1	346	0	479	0	289,956	274	406,866	112
		1	105	0	183	0	75	0	117	0	89	0	269	0
40	120	3	28,676	28	50,718	3	5,528	5	7,699	0	175,926	160	306,739	59
		5	489,239	471	742,088	180	6,546	6	8,576	0	643,193	569	793,226	1,394
		1	110	0	257	0	56	0	255	0	124	0	203	0
	240	3	152,076	139	281,777	16	117,397	106	160,612	12	5,279	5	7,620	1
		5	540,335	496	826,344	148	535,793	481	688,519	204	61,141	62	80,667	13
		3	32,167	103	51,040	4	1,471	4	2,014	0	40,850	139	60,342	10
	120	5	148,710	480	202,313	21	15,474	46	20,672	2	177,146	592	238,011	94
		10	928,120	2,507	1,221,123	320	1,341	4	2,684	0	857,098	2,268	1,099,466	1,036
		3	45,639	142	79,840	9	8,164	25	11,351	1	38,194	115	70,246	18
60	180	5	67,984	213	116,320	10	86,939	265	116,289	11	154,747	480	225,386	71
		10	554,179	1,606	709,248	135	2,899	8	4,074	0	659,057	1,802	826,902	676
		3	36,046	105	69,377	9	63,044	186	100,276	10	29,425	87	56,478	17
	360	5	65,693	191	122,537	11	260,475	748	347,087	61	116,561	336	187,802	80
		10	325,267	938	380,861	139	199,116	522	390,033	94	612,988	1,687	760,303	839
		3	21,956	113	31,955	4	5,548	26	7,968	1	24,212	129	33,813	11
	140	5	35,555	182	49,787	5	43,224	203	58,370	5	68,781	378	93,559	32
		10	433,861	1,896	561,273	80	64,858	261	96,471	9	478,454	2,012	614,807	287
		3	23,237	115	36,720	4	17,697	87	26,586	2	15,482	73	26,691	9
70	210	5	28,369	135	49,653	6	102,048	482	136,378	15	36,351	179	53,822	16
		10	199,631	957	250,188	47	425,445	1,723	591,432	321	241,655	1,094	302,173	154
		3	15,695	71	25,868	4	14,161	65	23,852	3	11,491	53	21,857	9
	420	5	31,448	143	60,485	7	110,057	499	151,761	21	43,205	194	76,126	29
		10	144,697	651	214,396	32	561,093	2,323	714,313	295	302,792	1,299	369,065	278
		5	2,881	43	3,528	1	65,948	915	88,633	20	3,712	59	4,596	3
	200	10	78,325	1,192	100,943	26	223,750	2,567	294,198	49	83,449	1,194	107,423	80
		15	245,786	2,908	314,423	66	366,963	4,129	570,878	192	218,288	2,528	279,448	171
		5	2,677	38	3,501	1	28,298	381	38,288	8	4,150	56	5,646	4
100	300	10	16,919	232	20,725	5	195,577	2,298	253,259	47	53,040	714	65,862	49
		15	73,470	1,000	88,731	21	359,932	4,105	478,912	108	152,474	1,845	189,237	140
		5	3,735	47	6,687	1	2,627	34	4,584	1	2,966	37	5,454	4
	600	10	11,999	153	22,451	4	63,551	813	77,867	21	49,280	609	63,494	50
		15	63,422	797	70,411	20	247,927	2,865	321,124	127	170,969	2,050	209,805	203

Table 10: Optimal solutions of the linear relaxations when detecting critical stars while minimizing the CNP objective. The values correspond to the results obtained with the original path constraints, the $\mathcal{T}$-path constraints, and the $\mathcal{T}$-path constraints strengthened with the 3-clique and 4-cycle inequalities. The ratios between such results are also presented.

Name	n	m	b	path	$\mathcal{T}$-path	$\mathcal{T}$-p-KC	$\frac{(\mathcal{T}\text{-path})-\text{path}}{\text{path}}\%$	$\frac{(\mathcal{T}\text{-p-KC})-(\mathcal{T}\text{-path})}{\mathcal{T}\text{-path}}\%$
high-tech-33	33	91	250	303.083	322.440	323.101	6.39	0.205
high-tech-33	33	91	350	224.025	247.000	247.571	10.26	0.231
high-tech-33	33	91	590	72.700	87.935	89.156	20.96	1.389
karate-34	34	78	250	182.098	187.692	187.692	3.07	0.000
karate-34	34	78	350	83.000	83.000	83.000	0.00	0.000
karate-34	34	78	590	20.520	20.520	20.520	0.00	0.000
kreb-62	62	153	350	567.200	567.200	567.200	0.00	0.000
kreb-62	62	153	590	306.592	312.900	312.900	2.06	0.000
kreb-62	62	153	1,200	59.850	61.929	61.929	3.47	0.000
dolphins-62	62	159	350	874.111	876.000	876.000	0.22	0.000
dolphins-62	62	159	590	551.670	607.372	607.372	10.10	0.000
dolphins-62	62	159	1,200	106.442	135.590	135.590	27.38	0.000
prison-67	67	142	350	1,303.145	1,387.840	1,387.840	6.50	0.000
prison-67	67	142	590	715.198	861.962	861.962	20.52	0.000
prison-67	67	142	1,250	103.154	123.828	123.828	20.04	0.000
huck-69	69	297	350	457.200	458.200	458.400	0.22	0.044
huck-69	69	297	590	255.816	255.816	255.816	0.00	0.000
huck-69	69	297	1,200	98.048	98.048	98.048	0.00	0.000
titanic-70	70	299	350	1,140.600	1,140.600	1,140.600	0.00	0.000
titanic-70	70	299	590	673.271	676.170	676.304	0.43	0.020
titanic-70	70	299	1,200	113.455	113.514	113.514	0.05	0.000
sanjuansur-75	75	144	350	1,651.034	1,745.650	1,745.650	5.73	0.000
sanjuansur-75	75	144	590	901.822	1,054.760	1,054.760	16.96	0.000
sanjuansur-75	75	144	1,200	155.014	193.556	193.556	24.86	0.000
jean-77	77	254	350	771.400	777.000	777.000	0.73	0.000
jean-77	77	254	590	356.400	357.827	357.827	0.40	0.000
jean-77	77	254	1,200	81.094	81.554	81.800	0.57	0.302
ca-sandi-86	86	124	350	305.75	305.750	305.750	0.00	0.000
ca-sandi-86	86	124	590	163.200	163.200	163.200	0.00	0.000
ca-sandi-86	86	124	1,200	44.867	45.857	45.857	2.21%	0.000
david-87	87	406	350	1,900.080	1,903.710	1,903.710	0.19	0.000
david-87	87	406	590	1,201.260	1,205.790	1,205.790	0.38	0.000
david-87	87	406	1,200	244.400	244.400	244.400	0.00	0.000
miles-92	92	327	590	1,016.970	1,072.470	1,072.510	5.46	0.004
miles-92	92	327	1,200	440.699	468.995	469.228	6.42	0.050
miles-92	92	327	2,400	101.361	126.300	126.704	24.60	0.320
forest-gump-94	94	271	590	114.960	115.800	115.800	0.73	0.000
forest-gump-94	94	271	1,200	60.400	61.489	61.489	1.80	0.000
forest-gump-94	94	271	1,750	34.000	38.697	38.697	13.81	0.000
ieeebus-118	118	179	590	1,210.290	1,326.550	1,326.550	9.61	0.000
ieeebus-118	118	179	1,200	365.583	415.000	415.000	13.52	0.000
ieeebus-118	118	179	1,750	180.392	200.667	200.667	11.24	0.000
anna-138	138	493	590	2,966.750	2,966.750	2,966.750	0.00	0.000
anna-138	138	493	1,200	422.275	422.275	422.275	0.00	0.000
anna-138	138	493	1,750	109.000	109.000	109.000	0.00	0.000
LindenStrasse-232	232	303	1,200	1,975.530	2,472.000	2,474.500	25.13	0.101
LindenStrasse-232	232	303	1,750	737.526	796.500	796.500	8.00	0.000
LindenStrasse-232	232	303	2,400	346.722	383.103	383.274	10.49	0.045

Table 11: Upper and lower bounds, and optimality gaps for detecting critical stars over the real-world graphs while minimizing the CNP objective.

		UB			LB			Gap %		
Name	b	path	$\mathcal{T}$-path	$\mathcal{T}$-p-KC	path	$\mathcal{T}$-path	$\mathcal{T}$-p-KC	path	$\mathcal{T}$-path	$\mathcal{T}$-p-KC
`high-tech-33`	250	351	351	351	351	351	351	0	0	0
`high-tech-33`	350	276	276	276	276	276	276	0	0	0
`high-tech-33`	590	162	162	162	162	162	162	0	0	0
`karate-34`	250	241	241	241	241	241	241	0	0	0
`karate-34`	350	83	83	83	83	83	83	0	0	0
`karate-34`	590	28	28	28	28	28	28	0	0	0
`kreb-62`	350	589	589	589	589	589	589	0	0	0
`kreb-62`	590	398	398	398	398	376	376	0	6	6
`kreb-62`	1,200	72	72	72	72	72	72	0	0	0
`dolphins-62`	350	876	876	876	876	876	876	0	0	0
`dolphins-62`	590	706	706	706	706	706	706	0	0	0
`dolphins-62`	1,200	**216**	219	224	139	**162**	**162**	55	**35**	38
`prison-67`	350	1,498	1,498	1,498	1,498	1,498	1,498	0	0	0
`prison-67`	590	1,109	1,109	1,109	**1,109**	1,050	**1,109**	**0**	6	**0**
`prison-67`	1,250	159	159	159	159	159	159	0	0	0
`huck-69`	350	485	485	485	485	485	485	0	0	0
`huck-69`	590	288	288	288	**288**	261	260	**0**	11	11
`huck-69`	1,200	103	103	103	103	103	103	0	0	0
`titanic-70`	350	1,177	1,177	1,177	1,177	1,177	1,177	0	0	0
`titanic-70`	590	844	844	844	**731**	720	712	**16**	17	18
`titanic-70`	1,200	139	**124**	**124**	119	**115**	**115**	17	**8**	**8**
`sanjuansur-75`	350	1,968	1,968	1,968	1,968	1,968	1,968	0	0	0
`sanjuansur-75`	590	1,433	1,433	1,433	**1,433**	1,391	1,378	**0**	3	4
`sanjuansur-75`	1,200	206	206	206	206	206	206	0	0	0
`jean-77`	350	820	820	820	820	820	820	0	0	0
`jean-77`	590	488	488	488	**488**	385	382	**0**	27	28
`jean-77`	1,200	88	88	88	**88**	86	**88**	**0**	2	**0**
`ca-sandi-86`	350	352	352	352	352	352	352	0	0	0
`ca-sandi-86`	590	179	179	179	179	179	179	0	0	0
`ca-sandi-86`	1,200	46	46	46	46	46	46	0	0	0
`david-87`	350	1,959	1,959	1,959	1,959	1,959	1,959	0	0	0
`david-87`	590	2,096	2,096	2,096	**1,309**	1,238	1,243	**60**	69	69
`david-87`	1,200	270	270	270	246	**244**	**244**	10	10	10
`miles-92`	590	1,206	1,206	1,206	**1,206**	1,140	1,140	**0**	6	6
`miles-92`	1,200	487	487	487	487	487	487	0	0	0
`miles-92`	2,400	195	177	177	114	**136**	**136**	71	**30**	**30**
`forest-gump-94`	590	135	135	135	**135**	123	122	**0**	10	10
`forest-gump-94`	1,200	66	66	66	66	66	66	0	0	0
`forest-gump-94`	1,750	**45**	46	**45**	**40**	41	**40**	14	**12**	**12**
`ieeebus-118`	590	1,666	1,666	1,666	**1,666**	1,585	1,576	**0**	5	6
`ieeebus-118`	1,200	444	444	444	444	444	444	0	0	0
`ieeebus-118`	1,750	206	206	206	206	206	206	0	0	0
`anna-138`	590	3,796	3,796	3,796	3,027	2,985	2,982	**25**	27	27
`anna-138`	1,200	538	468	**466**	425	424	424	27	**10**	**10**
`anna-138`	1,750	109	109	109	109	109	109	0	0	0
`LindenStrasse-232`	1,200	10,016	5,197	**4,345**	2,338	2,616	**2,633**	328	99	**65**
`LindenStrasse-232`	1,750	830	830	830	830	830	830	0	0	0
`LindenStrasse-232`	2,400	405	403	403	374	**403**	**403**	8	**0**	**0**

Table 12: Execution times for detecting critical stars over the real-world graphs while minimizing the CNP objective.

		Solution Time (s)			Separation Calls			Total Separation Time (s)		
Name	b	path	$\mathcal{T}$-path	$\mathcal{T}$-p-KC	path	$\mathcal{T}$-path	$\mathcal{T}$-p-KC	path	$\mathcal{T}$-path	$\mathcal{T}$-p-KC
high-tech-33	250	**18**	21	19	1,238	1,050	**913**	**1**	4	3
high-tech-33	350	17	**11**	15	902	**525**	618	**1**	2	2
high-tech-33	590	**583**	3,720	**3,972**	124,579	527,994	567,858	**67**	2,385	2,515
karate-34	250	3	3	3	537	**395**	442	**0**	1	2
karate-34	350	**0**	1	1	**13**	92	84	**0**	0	1
karate-34	590	**4**	12	11	**1,667**	4,616	3,989	**1**	6	5
kreb-62	350	**3**	9	10	**100**	194	201	**0**	6	7
kreb-62	590	**878**	7,200	7,200	**72,862**	228,983	226,309	**207**	5,985	5,907
kreb-62	1,200	135	**95**	118	22,179	**14,481**	17,858	61	**38**	49
dolphins-62	350	**5**	7	7	**47**	127	129	**0**	4	4
dolphins-62	590	2,814	**1,677**	1,754	97,801	51,388	**51,194**	**302**	705	743
dolphins-62	1,200	7,200	7,200	7,200	572,444	**312,057**	316,805	**1,722**	4,777	4,784
prison-67	350	272	325	**147**	3,209	4,259	**1,597**	**15**	83	31
prison-67	590	**4,130**	7,200	6,662	**88,773**	178,394	155,216	**396**	3,786	3,289
prison-67	1,250	2,467	**476**	602	180,829	**39,103**	52,300	675	**297**	402
huck-69	350	69	**39**	41	1,516	234	**231**	**6**	29	26
huck-69	590	**612**	7,200	7,200	**53,069**	73,255	67,106	**200**	6,613	6,373
huck-69	1,200	**264**	568	856	**32,452**	41,560	53,763	**121**	136	164
titanic-70	350	**8**	66	79	**130**	299	255	**1**	38	37
titanic-70	590	7,200	7,200	7,200	132,971	70,542	**66,460**	**554**	4,717	4,834
titanic-70	1,200	7,200	7,200	7,200	576,602	307,527	**234,024**	2,259	2,723	**2,052**
sanjuansur-75	350	464	415	**336**	4,748	**3,129**	3,416	**32**	83	86
sanjuansur-75	590	5,133	7,200	7,200	**101,947**	144,016	143,676	**640**	3,926	3,974
sanjuansur-75	1,200	331	**5**	13	16,020	182	758	87	1	6
jean-77	350	7	**52**	53	**146**	303	292	0	38	37
jean-77	590	**6,652**	7,200	7,200	265,047	84,614	**84,529**	**1,388**	6,114	6,067
jean-77	1,200	**6,159**	7,200	7,196	**793,123**	1,085,331	878,545	4,060	2,857	**2,291**
ca-sandi-86	350	**3**	13	11	127	521	457	**1 0.**	10	8
ca-sandi-86	590	26	**12**	**12**	2,210	1,054	1,169	15	**7**	**7**
ca-sandi-86	1,200	0	0	0	15	15	**13**	0	0	0
david-87	350	2,063	1,648	**1,385**	4,782	2,620	2,113	**42**	954	822
david-87	590	7,200	7,200	7,200	41,561	**13,673**	14,534	**367**	5,101	5,297
david-87	1,200	7,200	7,200	7,200	210,257	31,158	**25,184**	**1,578**	4,958	4,293
miles-92	590	**2,344**	7,200	7,200	**29,744**	57,419	54,361	**293**	4,625	4,641
miles-92	1,200	379	**112**	121	7,213	1,722	**1,655**	65	**35**	37
miles-92	2,400	7,200	7,200	7,200	423,838	**170,948**	171,373	3,757	**3,077**	3,124
forest-gump-94	590	**136**	7,200	7,200	7,940	39,044	44,069	**73**	6,306	6,316
forest-gump-94	1,200	**1,077**	2,043	2,480	75,069	**45,960**	56,804	679	**214**	265
forest-gump-94	1,750	7,200	7,200	7,200	552,366	149,815	**98,414**	4,994	700	**457**
ieeebus-118	590	1,726	7,200	7,200	**27,287**	94,024	91,595	**588**	4,774	5,285
ieeebus-118	1,200	261	**38**	53	7,391	**1,627**	2,279	135	**22**	32
ieeebus-118	1,750	165	15	**13**	5,897	792	729	**107**	9	7
anna-138	590	7,200	7,200	7,200	20,451	**6,077**	6,796	**634**	4,561	4,113
anna-138	1,200	7,200	7,200	7,200	91,843	**21,399**	23,892	**2,727**	5,302	4,901
anna-138	1,750	**4**	13	20	21	36	**20**	1	1	1
LindenStrasse-232	1,200	7,200	7,200	7,200	**9,926**	12,969	15,470	**1,729**	5,014	5,065
LindenStrasse-232	1,750	1,457	**569**	**643**	6,293	4,093	4,744	877	**387**	437
LindenStrasse-232	2,400	7,200	**4,213**	4,432	42,779	43,640	**41,725**	5,801	**3,188**	3,472