European Journal of Operational Research 307 (2023) 20–32

Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Discrete Optimization

On designing networks resilient to clique blockers

Haonan Zhong[a], Foad Mahdavi Pajouh[b,*], Oleg A. Prokopyev[c]

[a] Mechanical Engineering Department, City College, Kunming University of Science and Technology, Xishan, Kunming 650032, China
[b] School of Business, Stevens Institute of Technology, 1 Castle Point Terrace, Hoboken, NJ 07030, USA
[c] Department of Industrial Engineering, University of Pittsburgh, 3700 O'Hara Street, Pittsburgh, PA 15261, USA

ARTICLE INFO

Article history:
Received 19 January 2022
Accepted 13 September 2022
Available online 19 September 2022

Keywords:
Networks
Integer programming
Network design
Network resiliency
Vertex clique blockers

ABSTRACT

Robustness and vulnerability analysis of networked systems is often performed using the concept of vertex blockers. In particular, in the minimum cost vertex blocker clique problem, we seek a subset of vertices with the minimum total blocking cost such that the weight of any remaining clique in the interdicted graph (after the vertices are blocked) is upper bounded by some pre-defined parameter. Loosely speaking, we aim at disrupting the network with the minimum possible cost in order to guarantee that the network does not contain cohesive (e.g., closely related) groups of its structural elements with large weights; such groups are modeled as weighted cliques. In this paper, our focus is on designing networks that are resilient to clique blockers. Specifically, we construct additional connections (edges) in the network and our goal is to ensure (at the minimum possible cost of newly added edges) that the adversarial decision-maker (or the worst-case realization of random failures) cannot disrupt the network (namely, the weight of its cohesive groups) at some sufficiently low cost. The proposed approach is useful for modeling effective formation and preservation of influential clusters in networked systems. We first explore structural properties of our problem. Then, we develop several exact solution schemes based on integer programming and combinatorial branch-and-bound techniques. Finally, the performance of our approaches is explored in a computational study with randomly-generated and real-life network instances.

1. Introduction

The problem of designing systems that are resilient to adversarial attacks and random failures appears in a variety of research domains; see, e.g., Chen, Zheng, Veremyev, Pasiliao, & Boginski (2021); Haimes (2009); Holling (1996); Yodo & Wang (2016). Loosely speaking, the primary goal of this problem is to ensure that a system of interest is capable of maintaining all (or some of) its functionality after sustaining a certain degree of damage (disruption). Naturally, this problem is also studied in the network analysis literature, in particular, when the considered system is modeled as a graph (Sharkey, Nurre Pinkley, Eisenberg, & Alderson, 2021).

1.1. Blockers and the minimum cost vertex blocker clique problem

In the graph-theoretical models, disruptions are represented by removals of vertices and/or edges from a given graph. For example, in a telecommunication system, a failed (or jammed) connection between two nodes corresponds to an edge removal; similarly, a failed structural element (e.g., a router, a sensor) in the considered system corresponds to a vertex removal. In the related network analysis and combinatorial optimization literature, the main research focus is on:

(i) detecting a subset of vertices (or edges) of the smallest total cost whose removal (the other terms commonly used are *interdiction* and *blocking*) ensures that some *graph property of interest* (e.g., the maximum weight of a clique in a graph) is restricted by some upper (or lower) bounds given by a pre-defined parameter; or
(ii) identifying a subset of vertices (or edges) with their total cost upper bounded by some pre-defined parameter, whose deletion results in the largest change (either increase or decrease depending on the application context) of the *considered graph property*.

We note that these two problem types are naturally linked. In particular, the objectives in the problems of type (i) appear in the constraints of the corresponding problems of type (ii), and vice versa. The considered combinatorial optimization problems are

* Corresponding author.
E-mail addresses: 20210086@kust.edu.cn (H. Zhong), fmahdav1@stevens.edu (F. Mahdavi Pajouh), droleg@pitt.edu (O. A. Prokopyev).

https://doi.org/10.1016/j.ejor.2022.09.013
0377-2217/

known as the *minimum vertex/edge blocker* problem (Mahdavi Pajouh, 2019; Mahdavi Pajouh, Walteros, Boginski, & Pasiliao, 2015; Ries et al., 2010; Wei, Walteros, & Pajouh, 2021) and the *most vital (critical) vertices/edges* problem (Furini, Ljubi, Martin, & San Segundo, 2019; Veremyev, Boginski, & Pasiliao, 2014a; Veremyev, Prokopyev, & Pasiliao, 2014b; 2015; 2019), respectively.

A number of important network characterizations are explored in the context of vertex/edge blockers, such as the clique (Mahdavi Pajouh, 2019; Mahdavi Pajouh, Boginski, & Pasiliao, 2014; Tang, Richard, & Smith, 2016), independent set (Bazgan, Toubaline, & Tuza, 2011), dominating set (Mahdavi Pajouh et al., 2015), vertex cover (Bazgan et al., 2011), spanning tree (Bazgan, Toubaline, & Vanderpooten, 2013; Frederickson & Solis-Oba, 1999; Lin & Chern, 1993), pair-wise connectivity (Arulselvan, Commander, Elefteriadou, & Pardalos, 2009; Di Summa, Grosso, & Locatelli, 2011; 2012; Shen, Nguyen, Xuan, & Thai, 2012; Veremyev et al., 2014a; Veremyev et al., 2014b), shortest path (Israeli & Wood, 2002; Khachiyan et al., 2008; Schieber, Bar-Noy, & Khuller, 1995), matching (Zenklusen, 2010a), and maximum flow (Afshari Rad & Kakhki, 2017; Altner, Ergun, & Uhan, 2010; Ghare, Montgomery, & Turner, 1971; Wollmer, 1964; Wong et al., 2017; Wood, 1993; Zenklusen, 2010b) network properties. In this paper, we focus on the concept of a *weighted clique*.

Formally, let $G = (V, E)$ be a simple undirected graph, where $V = \{1, \ldots, n\}$ and $E \subseteq \{(i, j) : \ i \in V, \ \ j \in V\}$ are its sets of vertices and edges, respectively. For any subset of vertices $S \subseteq V$, let $G[S] = (S, \widetilde{E})$ denote the subgraph *induced* by S in G, where $\widetilde{E} = \{(i, j) \in E \ : \ i, j \in S\}$.

A *clique* C is a subset of V such that $G[C]$ is a complete graph; the problem of finding a clique of maximum cardinality (weight) in a given graph is referred to as the *maximum (weight) clique problem*. This problem is one of the classical *NP*-hard combinatorial optimization problems (Pardalos & Xue, 1994). There is a variety of important applications, where the concept of a clique is used to represent a cohesive (closely related) group of structural elements of a system modeled as a graph (Wu & Hao, 2015).

Assume that the graph is vertex-weighted, that is, we are given a vector of vertex weights $\mathbf{w} = (w_1, \ldots, w_n)^T$, where $w_i > 0$ for all $i \in V$. We denote by $W(Q)$ the weight of a set $Q \subseteq V$, which is simply the sum of the weights of all vertices in Q, i.e., $W(Q) = \sum_{i \in Q} w_i$.

Then, the *minimum cost vertex blocker clique (MCVBC)* problem is defined as follows (Mahdavi Pajouh et al., 2014; Nasirian, Mahdavi Pajouh, & Namayanja, 2019). Suppose that the decision-maker is allowed to block (remove) vertices from the graph and the corresponding blocking costs are given by $\mathbf{b} = (b_1, \ldots, b_n)^T$, where $b_i > 0$ for all $i \in V$. The decision-maker's goal is to find a subset of vertices $D \subseteq V$ with the minimum total blocking cost such that for any remaining clique, Q, in the interdicted graph $G[V \setminus D]$, its weight, $W(Q)$, is at most η, which is some given positive parameter. That is, the decision-maker solves the following problem:

$$[\text{MCVBC}]: \qquad \mathcal{B}(G, \eta) = \min_{D \subseteq V} \{B(D) \ : \ \omega^{\mathbf{w}}(G[V \setminus D]) \leq \eta\},$$

where $B(D) := \sum_{i \in D} b_i$ is the *total blocking cost* and $\omega^{\mathbf{w}}(G[V \setminus D])$ denotes the maximum weight of a clique in the interdicted network.

As a side note, we should mention about the *maximum clique interdiction problem* considered by Furini et al. (2019), which is closely related to the MCVBC problem. In particular, in the model by Furini et al. (2019), the objective is to minimize the size of a maximum clique in a graph subject to a budgetary restriction on the number of nodes blocked (interdicted); recall our earlier discussion on the two possible problem types and the links between them.

To summarize, when solving the MCVBC problem, we assume that G models an adversarial network. Then by solving the considered clique blocker problem our goal is to disrupt this network at the minimum possible vertex blocking cost to guarantee that the network does not contain cohesive (i.e., organized, pairwise connected, closely related) groups of structural elements that have their total vertex weight above some pre-defined parameter

1.2. Designing networks resilient to clique blockers

In this paper, our focus is on designing networks that are resilient to clique blockers. Specifically, assume that we are allowed to construct additional pairwise connections (i.e., edges) in the network at some cost given by $\mathbf{c} = (c_{ij} \ : \ (i, j) \in \check{E})^T$, where $\check{E}$ denotes a *candidate* set of non-adjacent vertex pairs. That is, $\check{E}$ is a subset of $\bar{E} = \{(i, j) \ : \ i, j \in V, \ \ (i, j) \notin E\}$, i.e., $\check{E} \subseteq \bar{E}$; set $\bar{E}$ is known to as the complement of E.

Denote by $\mathcal{C}(\hat{E}) := \sum_{(i,j) \in \hat{E}} c_{ij}$ the total cost of additional edges constructed in the designed network $\hat{G} = (V, E \cup \hat{E})$. Then the *clique-blocker-resilient network design (CBRND)* problem is given as follows:

$$[\text{CBRND}]: \quad \min_{\hat{E} \subseteq \check{E}} \left\{\mathcal{C}(\hat{E}) \ : \ \mathcal{B}(\hat{G}, \eta) \geq \beta, \ \text{where } \hat{G} = (V, E \cup \hat{E})\right\},$$

and β is an additional pre-defined positive parameter.

In other words, in the considered network design setting, we assume that there exists an adversarial decision-maker, who solves the minimum cost vertex blocker clique problem. Naturally, this decision-maker may also correspond to the worst-case realization of random failures in a given networked system modeled as G. We construct additional connections in the network, and our goal is to guarantee (at the minimum possible cost of newly added edges) that this adversarial decision-maker needs to have at least β units in its budget whenever he/she needs to ensure that the maximum weight of a clique in the interdicted network is at most η.

Loosely speaking, by adding edges to the network we provide a certain level of protection, which is given by some pre-defined parameters η and β in the considered problem setting. That is, the adversarial decision-maker needs to have sufficiently large interdiction (blocking) budget in order to disrupt the cohesive subgroups in the network.

Fig. 1 (a) illustrates an instance of the MCVBC problem, where $V = \{1, 2, 3, 4, 5\}$, $E = \{(1, 2), (1, 3), (2, 3), (2, 4)\}$, $\mathbf{w} = (1, 1, 1, 1, 2)^T$, $\mathbf{b} = (2, 1, 2, 2, 2)^T$, and $\eta = 2$. For this MCVBC problem instance, we have $\mathcal{B}(G, \eta) = 1$. Fig. 1(b) shows an instance of the CBRND problem with the same graph, where $\check{E} = \{(1, 4), (3, 4), (2, 5), (4, 5)\}$, $c_{14} = 1, \ c_{34} = 1, \ c_{25} = 1, \ c_{45} = 1$, and $\eta = \beta = 2$. An optimal solution for this CBRND problem is to construct edge (4,5); that is, $\hat{E} = \{(4, 5)\}$, which results in $\mathcal{B}(\hat{G}, \eta) = 3$.

1.3. Our contribution and the paper structure

To the best of our knowledge, our work is the first study on designing networks that are resilient to blockers, specifically, clique blockers. We address some basic structural properties of the problem (including a brief discussion on its computational complexity) in Section 2. Next, we contribute to the literature by providing an integer programming (IP) model, which can be used with a lazy-fashioned branch-and-cut (LBC) algorithm; see Section 3. As an alternative (and somewhat more advanced) solution scheme, we develop a combinatorial branch-and-bound (CBB) algorithm for our problem; see Section 4. The proposed solution approaches are then compared in an extensive computational study with randomly-generated and real-life network instances; see Section 5. Finally, we provide some concluding remarks and outline directions for future research in Section 6.

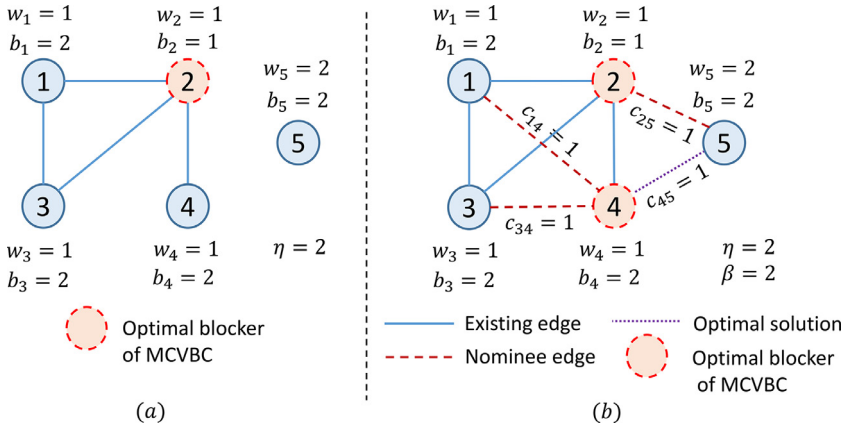

Fig. 1. (a) An instance of the minimum cost vertex blocker clique (MCVBC) problem with $\eta = 2$. (b) An instance of the clique-blocker-resilient network design (CBRND) problem on the same graph with $\eta = \beta = 2$.

2. Structural properties and computational complexity

We first outline some additional notations and assumptions made throughout the paper; see Section 2.1. Furthermore, as briefly outlined in the previous section, we describe some basic characterization of feasible solutions of the CBRND problem in Section 2.2 and address its theoretical computational complexity in Section 2.3. Note that the results from Section 2.2 provide the basis for establishing the correctness of the proposed IP model and our combinatorial based algorithms in Sections 3 and 4, respectively.

2.1. Additional notations, and assumptions

For simplicity, we use $P = (G, \check{E}, \mathbf{w}, \mathbf{b}, \mathbf{c}, \eta, \beta)$ to denote an instance of the CBRND problem. We assume that all components of $\mathbf{c}$ are strictly positive, while η and β are nonnegative. Also, we assume that all instances of the CBRND problem considered in this paper are feasible. It implies that $\hat{E} = \check{E}$ forms a feasible solution to the CBRND problem. That is, for each $D \subseteq V$ with $B(D) < \beta$, there exists a set $Q \subseteq V \setminus D$ such that Q is a clique in $\hat{G} = (V, E \cup \check{E})$ and $W(Q) > \eta$; consequently, we have $\beta \leq B(V)$.

Finally, we assume $w_i \leq \eta$ for all $i \in V$. Otherwise, a vertex with weight larger than η needs to be always blocked in every feasible solution to the CBRND problem.

2.2. Basic characterization of a feasible solution

We first introduce two concepts, i.e., *β-maximal-blockers* and *η-minimal-clique-candidates*, which are used to characterize the set of feasible solutions of our problem. These two concepts are defined as follows.

Definition 1. Given $P = (G, \check{E}, \mathbf{w}, \mathbf{b}, \mathbf{c}, \eta, \beta)$, a β-maximal-blocker is a subset of vertices $D \subseteq V$ such that $B(D) < \beta$ and D is not strictly contained in another subset of vertices satisfying this condition. The set containing all β-maximal-blockers is denoted by $\Phi^V_{\mathbf{b},\beta}$.

Definition 2. Given $P = (G, \check{E}, \mathbf{w}, \mathbf{b}, \mathbf{c}, \eta, \beta)$, an η-minimal-clique-candidate is a subset of vertices $Q \subseteq V$ such that $\{(i, j) : i, j \in Q\} \subseteq (E \cup \check{E})$, $W(Q) > \eta$, and Q does not strictly contain another subset of vertices satisfying these conditions. The set containing all η-minimal-clique-candidates is denoted by $\Psi^V_{\mathbf{w},\eta}$.

Feasibility of a solution can be verified using the concepts of β-maximal-blockers and η-minimal-clique-candidates as we formally establish next.

Proposition 1. *Given an instance $P = (G, \check{E}, \mathbf{w}, \mathbf{b}, \mathbf{c}, \eta, \beta)$, set $\hat{E} \subseteq \check{E}$ is a feasible solution of P if and only if for each $D \in \Phi^V_{\mathbf{b},\beta}$, there exists $Q \in \Psi^V_{\mathbf{w},\eta}$ such that $Q \cap D = \emptyset$ and Q is a clique in $\hat{G} = (V, E \cup \hat{E})$.*

Proof. $\Longrightarrow$ Let $\hat{E}$ be a feasible solution. Therefore, the cost of blocking cliques in $\hat{G}$ is at least β, i.e., $\mathcal{B}(\hat{G}, \eta) \geq \beta$. That is, regardless of how vertices in graph $\hat{G}$ are removed, if the total blocking cost is strictly less than β, then we can always find a clique of weight strictly greater that η in the remaining graph.

By Definition 1, the blocking costs of all β-maximal-blockers in $\Phi^V_{\mathbf{b},\beta}$ are strictly less than β. Thus, after removing a set $D \in \Phi^V_{\mathbf{b},\beta}$ from $\hat{G}$, there should exist $Q' \subseteq V \setminus D$ such that $W(Q') > \eta$ and Q' is a clique in $\hat{G}$. By Definition 2, Q' is either an η-minimal-clique-candidate or contains an η-minimal-clique-candidate. Thus, there exists $Q \in \Psi^V_{\mathbf{w},\eta}$ such that $Q \cap D = \emptyset$ and Q is a clique in $\hat{G}$.

$\Longleftarrow$ Consider an arbitrary set $D' \subseteq V$ with its blocking cost strictly less than β. By Definition 1, D' is either a β-maximal-blocker or is contained in a β-maximal-blocker. Therefore, there exists $D \in \Phi^V_{\mathbf{b},\beta}$ such that $D' \subseteq D$. By our assumption, we know that there exists $Q \in \Psi^V_{\mathbf{w},\eta}$ such that $Q \cap D = \emptyset$ and Q is a clique in $\hat{G} = (V, E \cup \hat{E})$. Also, $W(Q) > \eta$ by Definition 2. Then it is easy to verify that Q also exists after removing D'. Note that D' is selected arbitrarily. Thus, there exists a clique of weight strictly greater than η in the remaining graph when any vertex set with its blocking cost strictly less than β is removed from $\hat{G}$. Therefore, $\mathcal{B}(\hat{G}, \eta) \geq \beta$ and $\hat{E}$ is a feasible solution to P. □

Next, we derive some upper bounds on the running times required to compute all β-maximal-blockers and η-minimal-clique-candidates. Our proofs are constructive and we outline the corresponding algorithms.

Lemma 1. *Given an instance $P = (G, \check{E}, \mathbf{w}, \mathbf{b}, \mathbf{c}, \eta, \beta)$, sets $\Phi^V_{\mathbf{b},\beta}$ and $\Psi^V_{\mathbf{w},\eta}$ can be constructed in $O(n^\beta)$ and $O(\eta n^{\eta+1})$, respectively.*

Proof. Let η, β, and all elements of $\mathbf{w}$ and $\mathbf{b}$ be rational numbers. Then they can be multiplied by a sufficiently large constant to make them all integers. Hence, without loss of generality, in the remainder of the proof we assume that η, β, and all elements of $\mathbf{w}$ and $\mathbf{b}$ are strictly positive integers, which is a common assumption in the integer programming literature.

In view of the assumption above, it is relatively easy to verify that the cardinality of any β-maximal-blocker is at most $\beta - 1$. Next, we show that the cardinality of any η-minimal-clique-candidate is also bounded above by $\eta + 1$. Assume that it is not the case, i.e., there exists an η-minimal-clique-candidate

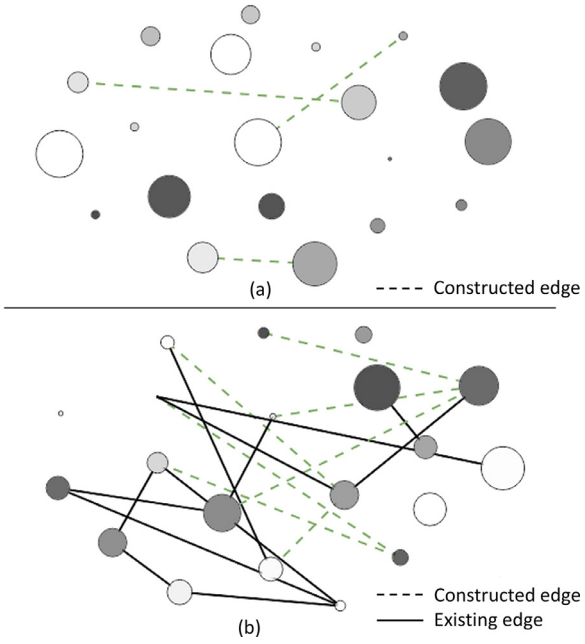

Fig. 2. (a) An optimal solution for the randomly-generated instance 20-10-10-4. (b) An optimal solution for the real-life network football-20-10-40. Vertices with higher weights are shown using darker colors, and vertices with higher blocking costs are larger in size.

$Q = \{i_1, \ldots, i_q\}$ such that $|Q| \geq \eta + 2$. By its definition, we know that $W(Q) = w_{i_1} + w_{i_2} + \ldots + w_{i_q} > \eta$. Also, from our earlier assumption, we have $|Q \setminus \{i_1\}| \geq \eta + 1 > \eta$. Recall that we assume that the elements of $\mathbf{w}$ are strictly positive integers. Then, $W(Q \setminus \{i_1\}) > \eta$, which implies that Q is not minimal and we have a contradiction; recall that Q is assumed to be minimal.

To construct set $\Phi^V_{\mathbf{b},\beta}$, one can employ a bounded depth-first-search enumeration, where each parent node is branched on to create as many children as there are vertices that are not already included into the parent's partially constructed β-maximal-blocker (which is initially empty at the root node). Naturally, the branching on each node is continued as long as the blocking cost of partially constructed β-maximal-blocker is less than β. If none of the children nodes of a parent node can be branched on, then the path connecting the root node to the parent node in the search tree corresponds to a β-maximal-blocker. Given the aforementioned upper bound on the cardinality of a β-maximal-blocker, the depth of the search tree is bounded by β (a depth of at most $\beta - 1$ for the β-maximal-blocker itself and one more layer containing children nodes that cannot be further branched on). Moreover, the number of tree nodes in each level is $O(n)$. Therefore, the outlined procedure can construct $\Phi^V_{\mathbf{b},\beta}$ in $O(n^\beta)$.

For $\Psi^V_{\mathbf{w},\eta}$, a similar approach can be used. Here, after including a vertex i into a partially constructed η-minimal-clique-candidate at a given search tree node, we need to make sure that all pairs of vertices in this partially constructed solution, that include vertex i belong to set $E \cup \check{E}$. Given the above mentioned upper bound on the size of a η-minimal-clique-candidate, this verification step can be done in $O(\eta)$ at each node of the search tree. The branching on each node is continued as long as the weight of the partially constructed η-minimal-clique-candidate is less than η. Whenever the weight of the partially constructed solution becomes at least η, the branching on the present tree node is terminated, and the constructed solution is examined to verify whether it satisfies the minimality condition in Definition 2. Again, using the upper bound on the size of a η-minimal-clique-candidate, this can be done in $O(\eta + 1)$ by finding and dropping the vertex with minimum weight from the obtained solution and verifying whether the weight of the remaining set is less than η. Given that the depth of the search tree in bounded by $\eta + 1$, and the number of tree nodes in each level is $O(n)$, we conclude that the running time needed to construct $\Psi^V_{\mathbf{w},\eta}$ is bounded by $O(\eta n^{\eta+1})$. □

2.3. On computational complexity

Define the decision version of the CBRND problem as follows. Given an instance $P = (G, \check{E}, \mathbf{w}, \mathbf{b}, \mathbf{c}, \eta, \beta)$ and $\alpha \geq 0$, the question is whether there exists a set $\hat{E} \subseteq \check{E}$ such that $\mathcal{B}(\hat{G}, \eta) \geq \beta$ and $\mathcal{C}(\hat{E}) \leq \alpha$, where $\hat{G} = (V, E \cup \hat{E})$.

The decision version of the classical maximum clique problem used in our reduction below, is given as follows. Given $G = (V, E)$ and an integer $k > 0$, we need to verify whether there exists a clique $D \subseteq V$ in G such that $|D| \geq k$. This problem is known to be *NP*-complete (Garey & Johnson, 1979)

Proposition 2. *The decision version of the CBRND problem is NP-hard.*

Proof. Given an instance of the maximum clique problem, consider the same graph G and let the candidate set $\check{E}$ be equal to the complement edge set $\bar{E}$. Furthermore, assume that the weight and the blocking cost of each vertex is equal to one, i.e., $w_i = b_i = 1$ for all $i \in V$. Similarly, the cost of adding an edge between any non-adjacent pair of vertices is also equal to one, i.e., $c_{ij} = 1$ for all $(i, j) \in \check{E}$.

Let $\eta = k - 1$. That is, the decision-maker in MCVBC aims at reducing the size of the maximum clique in G to at most $k - 1$. Next, observe that $\mathcal{B}(G, k-1) \geq 1$ if only if G contains a clique of size at least k. Therefore, by setting $\beta = 1$ and $\alpha = 0$, we reduce the problem of verifying whether there exists a clique of size at least k in G to a special case of the aforementioned decision version of the CBRND problem. Namely, G contains a clique of size at least k if only if we obtain a feasible solution to the CBRND problem by setting $\hat{E} = \emptyset$ with $\mathcal{C}(\hat{E}) = 0$. □

The above proof is relatively straightforward and we provide it to illustrate the fact that the considered CBRND problem is difficult from the theoretical perspective. However, one could argue that a somewhat stronger result could be established. We note that the decision version of the maximum clique interdiction problem is known to be Σ^p_2-complete; we refer to the discussion by Furini et al. (2019) and the earlier work by Rutenburg (1994). Recall that in the decision version of the maximum clique interdiction problem, we verify whether it is possible to remove a subset of at most β nodes from a given graph to ensure that the graph does not contain a clique of size η. From the proof of Proposition 2, we observe that by setting $\alpha = 0$, we reduce the CBRND problem to verifying whether $\mathcal{B}(G, \eta) \geq \beta$. The latter (assuming that the vertices blocking costs and weights are all ones) is equivalent to verifying whether the graph contains a clique of size $\eta + 1$ for any removal of at most $\beta - 1$ nodes. Therefore, one should expect that the CBRND problem should be Π^p_2-hard; see, e.g., Garey & Johnson (1979) for the definitions and other related complexity classes. Finally, we leave the question of pinpointing precise complexity of the CBRND problem as a direction for future research.

3. Integer programming (IP) techniques

Next, we describe an IP model for the CBRND problem. In particular, the results of Proposition 1 form the basis for this formulation. Consequently, we outline a lazy-fashioned branch-and-cut approach to solve the proposed IP model.

3.1. Base IP model

We first re-define the CBRND problem by employing the concepts of β-maximal-blockers and η-minimal-clique-candidates; recall Definitions 1 and 2, respectively. Specifically, given $S \subseteq \Psi^V_{\mathbf{w},\eta}$, we denote the unique set containing all elements of $\check{E}$ whose construction is necessary to transform all η-minimal-clique-candidates in S into cliques as $E(S)$. Furthermore, let $\widehat{\Phi}(S) \subseteq \Phi^V_{\mathbf{b},\beta}$ to be the collection of all β-maximal-blockers whose removal from $\hat{G} = (V, E \cup E(S))$ results in a graph in which the maximum weight of a clique is not greater than η.

In view of the above definitions, by Proposition 1, the CBRND problem is equivalent to

$$\min_{S \subseteq \Psi^V_{\mathbf{w},\eta}} \left\{ \mathcal{C}(E(S)) \ : \ \widehat{\Phi}(S) = \emptyset \right\},$$

which we next reformulate as an IP model.

We define two sets of binary variables, denoted by x_{ij} for all $(i,j) \in \check{E}$ and y_Q for all $Q \in \Psi^V_{\mathbf{w},\eta}$. Let $x_{ij} = 1$ if and only if an edge is constructed between vertices i and j, where $(i,j) \in \check{E}$. Also, let $y_Q = 1$ if and only if $Q \in \Psi^V_{\mathbf{w},\eta}$ is contained in a feasible solution of our model. Then, the CBRND problem can be formulated as follows:

$$\min \quad \sum_{(i,j) \in \check{E}} c_{ij} x_{ij}, \tag{1a}$$

$$\text{s. t.} \quad \sum_{Q \in \Psi^V_{\mathbf{w},\eta} \ : \ Q \cap D = \emptyset} y_Q \geq 1, \quad \forall D \in \Phi^V_{\mathbf{b},\beta}, \tag{1b}$$

$$|\{(i,j) \in \check{E} \ : \ i,j \in Q\}| y_Q \leq \sum_{(i,j) \in \check{E}, i,j \in Q} x_{ij}, \quad \forall Q \in \Psi^V_{\mathbf{w},\eta}, \tag{1c}$$

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

$$y_Q \in \{0,1\}, \quad \forall Q \in \Psi^V_{\mathbf{w},\eta}, \tag{1e}$$

where the objective function in (1a) minimizes the total edge construction cost. Constraint (1b) ensures that for each possible blocking of cost less than β, there exists a clique of weight larger than η in the remaining network. Constraint (1c) ensures that the vertices in Q (that correspond to $y_Q = 1$) form a clique in the designed network, i.e., there is a newly constructed edge between all disconnected node pairs in the original graph.

Finally, we note that the IP model given by (1a)–(1e) requires identification of all elements of sets $\Phi^V_{\mathbf{b},\beta}$ and $\Psi^V_{\mathbf{w},\eta}$. This step could take exponential time (with respect to β and η) as outlined in the proof of Lemma 1.

3.2. Lazy-fashioned branch-and-cut (LBC) algorithm

Here, we describe a lazy-fashioned branch-and-cut (LBC) algorithm for solving the IP model (1a)–(1e). As outlined earlier, this IP model requires enumeration of all β-maximal-blockers and η-minimal-clique-candidates. As shown by Lemma 1, constructing these sets is computationally expensive; recall that it might take exponential time with respect to β and η. Therefore, it might be beneficial to incorporate one or both of the corresponding constraint sets, see (1b) and (1c), as lazy cuts. However, we observe that enumerating all η-minimal-clique-candidates is somewhat inevitable as it is required for the definition of variables y_Q. Hence, we implement only constraints (1b) as lazy cuts.

We start the algorithm with a *relaxed master problem* (RMP), which is simply the IP model (1a)–(1e) without constraints (1b). We solve the RMP by using a standard branch-and-cut algorithm (e.g., via an off-the-shelf IP solver). Then, whenever a feasible solution to the RMP is returned at a search tree node, we solve a sub-problem (see below) to verify whether there exists a β-maximal-blocker that is "intersecting" all constructed η-minimal-clique-candidates.

That is, the blocker contains at least one node in common with every η-minimal-clique-candidate and (1b) is violated. If such β-maximal-blocker does not exist, then we can fathom the tree node as we have a feasible solution, and update the incumbent solution if necessary. Otherwise, we add constraint (1b) associated with the identified β-maximal-blocker to the RMP in order to cut off the current solution.

Specifically, denote a feasible solution to the RMP by $(\widehat{\mathbf{x}}, \widehat{\mathbf{y}})$. Recall that in the proof of Lemma 1, we assume that all parameters are integers. If this assumption does not hold, then there exists a strictly positive constant k such that η, β, and all elements of $\mathbf{w}$ and $\mathbf{b}$ become integers if they are multiplied by k.

Define a set of binary variables z_i, $i \in V$, where $z_i = 1$ if and only if i belongs to a β-maximal-blocker. Then, we consider the following feasibility problem:

$$\sum_{i \in V} k b_i z_i \leq k\beta - 1, \tag{2a}$$

$$\sum_{i \in Q} z_i \geq 1, \qquad \forall Q \in \Psi^V_{\mathbf{w},\eta} \text{ s.t. } \widehat{y}_Q = 1, \tag{2b}$$

$$z_i \in \{0,1\}, \qquad \forall i \in V, \tag{2c}$$

where constraint (2a) ensures that the blocking cost of set $D = \{i \in V \ : \ z_i = 1\}$ is strictly less than β, i.e., D defines an appropriate blocker. Constraint (2b) ensures that set D intersects all constructed η-minimal-clique-candidates.

Clearly, if (2a)–(2c) is infeasible, then $(\widehat{\mathbf{x}}, \widehat{\mathbf{y}})$ is a feasible solution of the CBRND problem. Otherwise, $(\widehat{\mathbf{x}}, \widehat{\mathbf{y}})$ is not a feasible solution of the CBRND problem as set D defines a blocker of cost smaller than β, that intersects all constructed η-minimal-clique-candidates. Note that D is not guaranteed to be a β-maximal-blocker. To find a β-maximal-blocker containing D, we recursively add vertices $i \in V \setminus D$ into D in the increasing order of b_i until adding one more vertex results in $B(D)$ being larger than or equal to β. Finally, constraint (1b) associated with the obtained β-maximal-blocker D, which is violated by $(\widehat{\mathbf{x}}, \widehat{\mathbf{y}})$, needs to be added into the RMP, and the algorithm continues.

4. Combinatorial branch-and-bound (CBB) algorithm

The focus of this section is on the combinatorial branch-and-bound (CBB) algorithm. Similar to the IP model in (1a)–(1e), the considered CBB algorithm is based on the characterization of feasible solutions given by Proposition 1. The structure of the search tree as well as the details of the upper and lower bounding approaches are described in Sections 4.1–4.3, respectively.

4.1. Search tree structure

We refer to a clique constructed by adding appropriate edges from $\check{E}$ to some η-minimal-clique-candidate, as an η-minimal-clique. Based on Proposition 1, the CBB algorithm constructs a collection of η-minimal-cliques with minimum construction costs such that for each β-maximal blocker, there exists an η-minimal-clique that does not have any vertex in common with the blocker. To enumerate all combinations of cliques constructed using η-minimal-clique-candidates, we apply a binary search tree, and split each tree node into two branches, namely, one associated

with constructing a clique using some chosen η-minimal-clique-candidate Q by adding all missing edges (to obtain a clique), and the other one related to not adding any additional edge between vertices in Q. The pseudo-code is given by Algorithm 1. Next, we provide its detailed discussion.

Denote by t^q, $q \geq 0$, the nodes of the search tree (or simply the tree in the discussion below), where q serves as an index of each node and t^0 corresponds to the root node of the tree. Each node t^q contains a *solution set* (denoted by S_q) and a *candidate set* (denoted by T_q). Specifically, set S_q is the collection of η-minimal-cliques constructed throughout the unique path connecting the root node t^0 and the current tree node t^q. Set T_q contains η-minimal-clique-candidates that have not been processed yet, and constructing a clique using them may result in a feasible solution to the CBRND problem.

To be more precise, given a tree node t^q, the set of η-minimal-clique-candidates that may result in a feasible solution to the CBRND problem is defined as

$$\widehat{\Psi}(S_q) = \left\{Q \in \Psi^V_{\mathbf{w},\eta} : \ Q \cap D = \emptyset \text{ for some } D \in \widehat{\Phi}(S_q)\right\}, \tag{3}$$

where we need to recall our discussion at the beginning of Section 3.1 for the definition of set $\widehat{\Phi}(S_q)$. Thus, set T_q contains elements of $\widehat{\Psi}(S_q)$ that belong to the candidate set of the parent node of node t^q and have not been chosen for branching yet. At the root node (i.e., $q = 0$), we have $S_0 = \emptyset$ and $T_0 = \widehat{\Psi}(S_0)$.

When branching on a parent node t^q, we first select an element Q of T_q as the branching variable, and then create two child nodes. Namely, the first one is based on constructing a clique using Q by adding the missing edges, and the other is based on not adding any additional edge between vertices in Q.

We note that S_q is a feasible solution if and only if $\widehat{\Phi}(S_q)$ is empty. Hence, the effectiveness of constructing a clique using a set $Q \in T_q$ can be measured by the decrease in the cardinality of $\widehat{\Phi}(S_q)$ after adding Q to S_q, i.e.,

$$\Delta\widehat{\Phi}_q(Q) = |\widehat{\Phi}(S_q)| - |\widehat{\Phi}(S_q \cup \{Q\})|.$$

Note that, when adding Q to S_q, some edges in $E(\{Q\})$ may have already been constructed as part of $E(S_q)$; recall the definition of $E(\cdot)$ at the beginning of Section 3.1. Thus, the additional cost of constructing a clique on Q is given by

$$\Delta\mathcal{C}_q(Q) = \mathcal{C}(E(\{Q\}) \setminus E(S_q)). \tag{4}$$

Consequently, the *branching variable selection rule* used here is to choose an element $Q \in T_q$ that has the highest effect per unit increase in its construction cost, i.e., the maximum ratio of $\Delta\widehat{\Phi}_q(Q)$ and $\Delta\mathcal{C}_q(Q)$. This branching variable selection rule has a good potential to find low-cost feasible solutions within a reasonably small number of iterations, which, in turn, decreases the depth of the search tree.

We create the child node associated with constructing a clique using η-minimal-clique-candidate Q prior to the child node corresponding to not adding any additional edge to set Q. For the first child node (denoted by t^p), the solution set is formed by adding set Q to set S_q, i.e., $S_p = S_q \cup \{Q\}$. The candidate set of this child node is composed of η-minimal-clique-candidates in $T_q \setminus \{Q\}$, which have the potential to make $E(S_p)$ feasible, i.e., $T_p = (T_q \setminus \{Q\}) \cap \widehat{\Psi}(S_p)$. The second child (denoted by t^r) has the same solution set as its parent node, i.e. $S_r = S_q$. Furthermore, for the candidate set, we simply need to set $T_r = T_q \setminus \{Q\}$. The detailed pseudo-code of outlined branching algorithm is given in Algorithm 2.

The search strategy used is a depth-first-search (DFS) strategy in which we choose an unprocessed node from the deepest layer of the tree for further processing. In case there are two nodes in the deepest layer, the node associated with constructing an η-minimal-clique is selected. This search strategy is consistent with our branching variable selection rule discussed above (selecting Q with maximum $\Delta\widehat{\Phi}_q(Q)/\Delta\mathcal{C}_q(Q)$), which encourages the detection of good quality feasible solutions (upper bounds) at the early stages of the algorithm.

Furthermore, it should be noted that if a tree node t^q associated with a clique construction is selected for further processing, then we first need to verify whether $E(S_q)$ is a feasible solution. If $\widehat{\Phi}(S_q)$ is empty, then $E(S_q)$ is a feasible solution according to Proposition 1. We then fathom node t^q by feasibility, and update the incumbent objective if necessary. If $E(S_q)$ is not a feasible solution, then we verify whether the lower bound at node t^q (discussed in Section 4.3) is smaller than the incumbent objective. If the answer is no, then we fathom node t^q by bound. Otherwise, the branching function is called (see Algorithm 2) and the children nodes of node t^q are created and added to the search tree.

Algorithm 1: A combinatorial branch-and-bound (CBB) algorithm.

Input: $P = (G, \check{E}, \mathbf{w}, \mathbf{b}, \mathbf{c}, \eta, \beta)$
Output: A set $E^* \subseteq \check{E}$ that is an optimal solution for the CBRND problem, i.e., $\mathcal{C}(E^*)$ is minimum possible given that for graph $G^* = (V, E \cup E^*)$, we have $\mathcal{B}(G^*, \eta) \geq \beta$

Node_Counter $\leftarrow 0$;
$S_0 \leftarrow \emptyset$; $T_0 \leftarrow \widehat{\Psi}(S_0)$; $t^0 \leftarrow (S_0, T_0)$; `// Create root node` t^0
Tree $\leftarrow \emptyset$; `// Initialize the search tree`
$E^* \leftarrow$ UB($\mathbf{c}$, $\Phi^V_{\mathbf{b},\beta}$, $\Psi^V_{\mathbf{w},\eta}$); `// See Section~4.2`
BRANCH($\mathbf{c}$, $\Phi^V_{\mathbf{b},\beta}$, $\Psi^V_{\mathbf{w},\eta}$, t^0, *Tree*, *Node_Counter*);
while *Tree* $\neq \emptyset$ **do**
  Select a node $t^q \in$ *Tree* using DFS strategy; `// See Section~4.1`
  Tree $\leftarrow$ *Tree* $\setminus \{t^q\}$;
  if t^q *is associated with constructing an* η*-minimal-clique* **then**
    if $E(S_q)$ *is feasible (i.e.,* $\widehat{\Phi}(S_q) = \emptyset$*)* **then** `// See Section~4.1`
      if $\mathcal{C}(E^*) > \mathcal{C}(E(S_q))$ **then**
        $E^* \leftarrow E(S_q)$; `// Update the incumbent solution`
      end
    end
    else
      if $\mathcal{C}(E^*) >$ LB($\mathbf{c}$, $\Phi^V_{\mathbf{b},\beta}$, $\Psi^V_{\mathbf{w},\eta}$, t^q) **then** `//Otherwise, fathom~`t^q
        BRANCH($\mathbf{c}$, $\Phi^V_{\mathbf{b},\beta}$, $\Psi^V_{\mathbf{w},\eta}$, t^q, *Tree*, *Node_Counter*);
      end
    end
  end
  else `//` t^q `is associated with not adding any additional edge`
    if $E(S_q) \cup E(T_q)$ *is feasible (i.e.,* $\widehat{\Phi}(S_q \cup T_q) = \emptyset$*)* **then**
      `// Otherwise, fathom` t^q `by infeasibility`
      if $\mathcal{C}(E^*) >$ LB($\mathbf{c}$, $\Phi^V_{\mathbf{b},\beta}$, $\Psi^V_{\mathbf{w},\eta}$, t^q) **then** `//Otherwise, fathom` t^q
        BRANCH($\mathbf{c}$, $\Phi^V_{\mathbf{b},\beta}$, $\Psi^V_{\mathbf{w},\eta}$, t^q, *Tree*, *Node_Counter*);
      end
    end
  end
end
return E^*;

Algorithm 2: The branching function.

Function BRANCH($\boldsymbol{c}$, $\Phi^V_{\boldsymbol{b},\beta}$, $\Psi^V_{\mathbf{W},\eta}$, t^q,*Tree*,*Node_Counter*)
- Select $Q \in T_q$ with the max value of $\Delta\widehat{\Phi}_q(Q)/\Delta\mathcal{C}_q(Q)$;
- $T_q \leftarrow T_q \setminus \{Q\}$;
- *Node_Counter* $\leftarrow$ *Node_Counter*+1;
- $p \leftarrow$ *Node_Counter*;
- `// Create` t^p `associated with adding` Q
- $S_p \leftarrow S_q \cup \{Q\}$;
- $T_p \leftarrow \widehat{\Psi}(S_p) \cap T_q$; `// See Section~4.1`
- $t^p \leftarrow (S_p, T_p)$;
- *Tree* $\leftarrow$ *Tree* $\cup \{t^p\}$;
- *Node_Counter* $\leftarrow$ *Node_Counter*+1;
- $r \leftarrow$ *Node_Counter*;
- `// Create` t^r `associated with NOT adding` Q
- $S_r \leftarrow S_q$;
- $T_r \leftarrow T_q$;
- $t^r \leftarrow (S_r, T_r)$;
- *Tree* $\leftarrow$ *Tree* $\cup \{t^r\}$;

end

Finally, node t^q, which corresponds to a clique construction, may never be fathomed by infeasibility; recall that the parent of node t^q is not fathomed by infeasibility. On the other hand, if t^q is associated with not adding any additional edge to some η-minimal-clique-candidate, then we need to verify whether $E(S_q) \cup E(T_q)$ is a feasible solution, which is done by checking whether $\widehat{\Phi}(S_q \cup T_q)$ is empty. If $\widehat{\Phi}(S_q \cup T_q)$ is not empty, then we fathom t^q by infeasibility. Otherwise, we continue to verify whether t^q can be fathomed by bound; if not, then the algorithm continues with branching.

4.2. Upper-bounding approach

We use a heuristic approach to find a feasible solution, and initialize the upper bound. The heuristic recursively constructs η-minimal-cliques in a greedy routine until the construction is feasible; see the pseudo-code in Algorithm 3.

Algorithm 3: An algorithm for finding a valid upper bound.

Function UB($\boldsymbol{c}$, $\Phi^V_{\boldsymbol{b},\beta}$, $\Psi^V_{\mathbf{W},\eta}$)
- $S^* \leftarrow \emptyset$;
- $T^* \leftarrow \widehat{\Psi}(S^*)$;
- **while** $E(S^*)$ *is not feasible (i.e.,* $\widehat{\Phi}(S^*) \neq \emptyset$*)* **do**
  - Pick $Q \in T^*$ with the maximum value of $|\Phi^V_{\mathbf{b},\beta} \setminus \widehat{\Phi}(\{Q\})|/\mathcal{C}(E(\{Q\}))$;
  - $S^* \leftarrow S^* \cup \{Q\}$;
  - $T^* \leftarrow \widehat{\Psi}(S^*)$; `// See Section~4.2`
- **end**
- **return** $E(S^*)$;

end

Specifically, we start with an empty solution set S^* and a candidate set T^*, which is initialized with $T^* = \widehat{\Psi}(S^*)$; recall Eq. (3) in Section 4.1. In each iteration, we find the η-minimal-clique-candidate $Q \in T^*$ with maximum value of $|\Phi^V_{\mathbf{b},\beta} \setminus \widehat{\Phi}(\{Q\})|/\mathcal{C}(E(\{Q\}))$ in T^*. Notice that we do not use $|\widehat{\Phi}(S^*) \setminus \widehat{\Phi}(S^* \cup \{Q\})|/\mathcal{C}(E(\{Q\}) \setminus E(S^*))$ in order not to update the ratio whenever an element is added to S^*. Then, we add Q to S^* and update T^* with η-minimal-clique-candidates in $T^* \setminus \{Q\}$ that have potential to make $E(S^*)$ feasible. We repeat this cycle until set $E(S^*)$ becomes a feasible solution.

The running time of the proposed upper bounding heuristic is exponential in the worst case as discussed in Appendix A.

4.3. Lower-bounding approach

Next, we discuss the approach for computing a valid lower bound at each node t^q of the search tree. Proposition 3 below presents one such bound using (4).

Proposition 3. *Given a tree node t^q, if $\hat{E} \subseteq \check{E}$ is a feasible solution to the CBRND problem that is within the subtree rooted at t^q, then*

$$\mathcal{C}(\hat{E}) \geq \mathcal{C}(E(S_q)) + \max\left\{\delta(D) \ : \ D \in \widehat{\Phi}(S_q)\right\},$$

where

$$\delta(D) = \min\{\Delta\mathcal{C}_q(Q) \ : \ Q \in T_q, \ Q \cap D = \emptyset\}.$$

Proof. The proof of Proposition 3 is provided in Appendix B. □

To use the lower bound from Proposition 3 at a node t^q, we need to compute the maximum value of $\delta(D)$ across all β-maximal-blockers $D \in \widehat{\Phi}(S_q)$, and then add this maximum value to the cost of $E(S_q)$, i.e., $\mathcal{C}(E(S_q))$. To find $\delta(D)$ for a given $D \in \widehat{\Phi}(S_q)$, we need to compute the minimum value of $\Delta\mathcal{C}_q(Q)$ across all $Q \in T_q$ that satisfy $Q \cap D = \emptyset$. The details of our algorithm are in Algorithm 4.

Algorithm 4: An algorithm for finding a valid lower bound at each search tree node.

Function LB($\boldsymbol{c}$, $\Phi^V_{\boldsymbol{b},\beta}$, $\Psi^V_{\mathbf{W},\eta}$, t^q)
- $\mathcal{C}_{\max} \leftarrow 0$;
- **foreach** $D \in \widehat{\Phi}(S_q)$ **do**
  - $\mathcal{C}_{\min} \leftarrow \infty$;
  - **foreach** $Q \in T_q$ *such that* $Q \cap D = \emptyset$ **do**
    - $\mathcal{C}_{\min} \leftarrow \min\{\mathcal{C}_{\min}, \Delta\mathcal{C}_q(Q)\}$; `// See Section~4.3`
  - **end**
  - $\mathcal{C}_{\max} \leftarrow \max\{\mathcal{C}_{\max}, \mathcal{C}_{\min}\}$;
- **end**
- **return** $\mathcal{C}(E(S_q)) + \mathcal{C}_{\max}$;

end

Finally, the running time for the proposed lower bounding procedure is also worst-case exponential. Set $\widehat{\Phi}(S_q)$ at t^q can be found by updating the corresponding set in the parent node of t^q. As discussed in Appendix A, this updating can be completed in $O(\eta\beta n^{k\beta-1})$. For a given $D \in \widehat{\Phi}(S_q)$ and $Q \in T_q$, verifying whether $Q \cap D = \emptyset$ and computing $\Delta\mathcal{C}_q(Q)$ both can be done in $O(\eta\beta + \eta^2 n^2)$. Hence, both for loops in Algorithm 4 can be computed in $O(n^{k\beta-1} n^{k\eta+1}(\eta\beta + \eta^2 n^2))$.

5. Computational study

Next, we study the computational performance of our approaches outlined in Sections 3 and 4. Specifically, we compare a standard branch-and-cut (BC) algorithm solving the IP formulation presented in Section 3.1, the LBC algorithm proposed in Section 3.2, and the CBB algorithm developed in Section 4. The first two approaches are implemented using Gurobi Optimizer 9.1.1. Also, it is important to point out that all three algorithms rely on the approaches outlined in the proof of Lemma 1 for computing sets $\Phi^V_{\mathbf{b},\beta}$ and/or $\Psi^V_{\mathbf{W},\eta}$; see additional discussion on this issue in Section 5.2.

All algorithms are coded in C++ and the numerical experiments are conducted on a 64-bit Windows system with Intel(R) Core(TM) i7-10750H processors and 16GB RAM. However, only one core is

Table 1
Results for the randomly-generated network instances (denoted as $20-x-y-z$). The running times are in seconds (Time). The optimality gaps (Gap), and the costs of the best solutions found (Solution) are also reported for each solution approach. The smallest run time (or the optimality gap) for each instance is highlighted and underlined.

Instance	BC			LBC			CBB		
	Time (seconds)	Gap (%)	Solution	Time (seconds)	Gap (%)	Solution	Time (seconds)	Gap (%)	Solution
20-10-10-0	0.237	0.000	0.078	0.369	0.000	0.078	**0.179**	0.000	0.078
20-10-10-1	0.097	0.000	0.555	0.210	0.000	0.555	**0.095**	0.000	0.555
20-10-10-2	**0.114**	0.000	0.430	0.225	0.000	0.430	0.134	0.000	0.430
20-10-10-3	**0.172**	0.000	0.454	0.277	0.000	0.454	0.205	0.000	0.454
20-10-10-4	0.211	0.000	0.539	**0.144**	0.000	0.539	0.189	0.000	0.539
20-10-40-0	307.864	0.000	18.590	**16.506**	0.000	18.590	TL	0.776	19.562
20-10-40-1	14.340	0.000	40.385	**3.156**	0.000	40.385	TL	0.808	45.851
20-10-40-2	34.476	0.000	38.355	**3.412**	0.000	38.355	TL	0.786	42.540
20-10-40-3	14.093	0.000	18.974	**0.806**	0.000	18.974	TL	0.823	20.199
20-10-40-4	95.025	0.000	24.187	**9.454**	0.000	24.187	TL	0.782	25.643
20-10-50-0	3209.020	0.000	101.761	**145.267**	0.000	101.761	TL	0.841	114.620
20-10-50-1	437.851	0.000	76.963	**29.491**	0.000	76.963	TL	0.822	95.659
20-10-50-2	58.785	0.000	45.341	**11.494**	0.000	45.341	TL	0.891	49.212
20-10-50-3	132.758	0.000	38.237	**3.784**	0.000	38.237	TL	0.862	45.060
20-10-50-4	11.291	0.000	41.809	**1.366**	0.000	41.809	TL	0.835	52.778
20-10-75-0	**71.019**	0.000	380.640	828.603	0.000	380.640	TL	0.895	445.530
20-10-75-1	7.258	0.000	429.103	**2.860**	0.000	429.103	TL	0.948	481.670
20-10-75-2	**292.250**	0.000	394.853	1617.640	0.000	394.853	TL	0.948	437.417
20-10-75-3	**287.860**	0.000	537.070	711.875	0.000	537.070	TL	0.879	600.757
20-10-75-4	**85.752**	0.000	267.099	410.252	0.000	267.099	TL	0.943	326.197
20-40-10-0	TL	0.937	158.745	TL	0.898	158.745	TL	**0.466**	190.729
20-40-10-1	TL	**0.364**	140.028	TL	0.979	140.028	TL	0.524	172.612
20-40-10-2	TL	**0.138**	89.249	TL	0.500	89.249	TL	0.534	125.496
20-40-10-3	TL	**0.553**	127.303	TL	0.635	127.303	TL	0.547	127.303
20-40-10-4	**83.142**	0.000	86.286	240.061	0.000	86.286	TL	0.383	88.085
20-40-40-0	PTL	–	–	TL	**0.368**	882.746	PTL	–	–
20-40-40-1	MTL	–	–	TL	**0.594**	951.690	TL	0.789	951.690
20-40-40-2	TL	0.810	997.458	TL	**0.666**	997.458	TL	0.689	898.587
20-40-40-3	TL	**0.600**	912.800	TL	0.626	983.726	TL	0.817	983.726
20-40-40-4	TL	0.598	807.051	TL	**0.564**	908.662	TL	0.686	961.118
20-50-10-0	TL	0.759	229.738	TL	0.666	229.738	TL	**0.455**	251.065
20-50-10-1	TL	0.955	352.274	TL	0.988	814.072	TL	**0.404**	412.213
20-50-10-2	TL	0.953	288.528	TL	0.978	411.824	TL	**0.524**	337.079
20-50-10-3	TL	0.965	277.450	TL	0.999	352.735	TL	**0.424**	295.317
20-50-10-4	TL	**0.449**	237.971	TL	1.000	606.736	TL	0.552	296.807
20-75-10-0	**4.321**	0.000	770.561	57.497	0.000	770.561	TL	0.275	772.175
20-75-10-1	**28.377**	0.000	612.303	499.071	0.000	612.303	TL	0.223	612.303
20-75-10-2	**28.958**	0.000	667.458	301.406	0.000	667.458	TL	0.297	673.206
20-75-10-3	**49.761**	0.000	638.295	1076.020	0.000	638.295	TL	0.353	646.119
20-75-10-4	**3.240**	0.000	826.781	8.424	0.000	826.781	TL	0.372	847.161

applied in all our experiments in order to have a fair comparison between the CBB algorithm and the other two integer programming based methods by not allowing parallelization. The run time limit is set to 3600 seconds for each problem instance.

5.1. Test instances

In our preliminary experiments, we observe that (as expected) increasing $|V|$ makes the problem more challenging for all algorithms, and does not affect their relative performance. Similarly (and also not surprisingly), as edge set E becomes larger and/or the nominee set $\check{E}$ becomes smaller, the instances become easier for all algorithms.

In view of the above observations, we fix the size of all instances to some value (namely, 20) for which at least one of our algorithms can find a feasible solution. For the randomly-generated instances, we set $E = \emptyset$. For the real-life instances, we consider five real-life networks chosen from different categories of Network Repository (Ryan & Nesreen, 2015), and pick 20 vertices with the smallest degrees in these networks. Thus, we ensure that the edge density among these 20 vertices is relatively low, and the problem of adding edges between these vertices to increase the network resiliency becomes more challenging and also more interesting from the practical perspective. Finally, we let $\check{E} = \bar{E}$ in all our instances.

The name of each randomly-generated instance is in the form $20-x-y-z$, see Tables 1 and 2; the name of each real-life instance is in the form $name-20-x-y$, see Tables 3 and 4; recall that for all instances $|V| = 20$. The blocking costs, weights, and connecting costs in all instances are positive rational numbers that are randomly generated between 0 and 10 using a uniform distribution. Parameters η and β are selected as $x\%$ and $y\%$ of the total sum of weights and blocking costs of all vertices, respectively. By adjusting x and y, we generate instances with different cardinalities of $\Psi^V_{\mathbf{w},\eta}$ and $\Phi^V_{\mathbf{b},\beta}$. The problem difficulty is closely related to the cardinalities of these two sets as we further highlight in Section 5.2.

For randomly-generated instances, combinations of x and y are chosen from set $\{10, 40, 50, 75\}$, and for each combination of these two parameters, we generate five random instances denoted by $z \in \{0, \ldots, 4\}$. Note that we can not generate random instances for all combinations of x and y in this set, as it is unlikely to find a feasible instance with large values of x and y simultaneously. After creating a randomly-generated instance for a given combination of x and y, if the instance is either infeasible or trivial (the latter case arises when not adding any edge is a feasible solution), then we discard it, and regenerate a new one. We repeat this procedure for each combination at most 10,000 times. If we can not generate five meaningful random instances for a given combination within

Table 2
Results for the randomly-generated network instances (denoted as $20-x-y-z$). The numbers of η-minimal-clique-candidates ($|\Psi^V_{\mathbf{w},\eta}|$) and β-maximal-blockers ($|\Phi^V_{\mathbf{b},\beta}|$) for each instance are reported. The preprocessing times are in seconds (P_T). We also report the number of the search tree nodes (# node) for each algorithm, the number of lazy cuts (# cut) applied by the LBC algorithm, and the initial gap (Gap^0 (%)) of the CBB algorithm.

Instance	$\lvert\Psi^V_{\mathbf{w},\eta}\rvert$	$\lvert\Phi^V_{\mathbf{b},\beta}\rvert$	BC		LBC			CBB		
			P_T (seconds)	# node	P_T (seconds)	# node	# cut	P_T (seconds)	# node	Gap^0 (%)
20-10-10-0	257	142	0.109	0	0.014	0	6	0.124	3	0.000
20-10-10-1	219	60	0.052	0	0.013	1	7	0.059	9	0.421
20-10-10-2	222	65	0.055	1	0.010	1	9	0.063	11	0.236
20-10-10-3	196	97	0.058	1	0.009	1	12	0.064	15	0.423
20-10-10-4	182	164	0.078	0	0.009	1	8	0.085	7	0.216
20-10-40-0	403	17,497	14.552	280	0.011	1902	395	14.982	95,013	0.949
20-10-40-1	179	11,470	4.984	1	0.009	642	287	5.923	897,669	0.959
20-10-40-2	241	10,299	6.242	117	0.009	764	221	6.462	469,653	0.984
20-10-40-3	209	14,142	6.895	1	0.010	178	127	7.740	187,435	0.925
20-10-40-4	325	14,427	10.457	126	0.012	1243	290	10.963	114,869	0.934
20-10-50-0	314	27,223	20.023	6303	0.014	18,256	366	21.245	893,107	0.980
20-10-50-1	259	21,701	13.931	2070	0.011	5711	454	14.939	1,541,803	0.978
20-10-50-2	225	21,479	12.063	171	0.010	2457	508	13.520	426,333	0.961
20-10-50-3	231	28,360	15.290	1	0.011	666	277	16.629	378,729	0.958
20-10-50-4	196	9981	6.445	1	0.009	347	216	7.145	1,233,929	0.968
20-10-75-0	384	5966	9.177	3064	0.013	239,377	895	9.793	3,765,677	0.947
20-10-75-1	240	3066	6.672	1	0.009	597	273	7.383	7,576,069	0.965
20-10-75-2	336	7618	9.764	9130	0.011	408,795	541	10.393	5,933,067	0.977
20-10-75-3	373	8743	11.033	13,344	0.015	280,421	1184	11.445	7,425,801	0.928
20-10-75-4	291	5277	8.274	2215	0.009	75,972	1013	8.678	3,163,763	0.979
20-40-10-0	36,487	75	12.251	527	2.311	799	26	15.232	19,623	0.580
20-40-10-1	21,697	158	13.912	10,261	1.568	879	31	15.978	30,033	0.671
20-40-10-2	12,359	76	4.659	4895	1.121	2763	17	9.072	46,495	0.698
20-40-10-3	18,183	102	7.428	104,425	1.302	1949	31	9.627	4529	0.752
20-40-10-4	25,021	93	8.521	313	1.536	2262	18	11.554	40,399	0.744
20-40-40-0	39,586	–	–	–	2.307	2941	2673	–	–	–
20-40-40-1	27,015	34,399	1928.840	–	1.872	641	209	1983.550	5	0.789
20-40-40-2	18,461	40,994	1534.870	0	1.797	573	225	1577.580	5	0.689
20-40-40-3	32,448	22,128	1463.770	1	2.005	566	118	1516.940	5	0.817
20-40-40-4	42,468	28,534	2377.61	0	2.486	543	163	2405.67	5	0.686
20-50-10-0	24,397	85	8.521	11,089	1.785	53,468	23	12.460	20,445	0.804
20-50-10-1	63,418	95	21.912	510	4.102	2552	49	33.994	7599	0.745
20-50-10-2	26,101	112	14.055	522	2.294	970	40	18.146	25,023	0.714
20-50-10-3	60,694	95	20.428	510	4.087	2196	37	30.201	14,609	0.673
20-50-10-4	31,702	101	13.493	5101	2.466	2358	32	18.781	9609	0.798
20-75-10-0	3647	81	1.748	1	0.496	156	41	2.900	164,741	0.425
20-75-10-1	6105	64	2.508	1	0.724	1076	50	4.559	162,411	0.443
20-75-10-2	3475	119	2.179	1	0.500	562	60	3.187	582,389	0.457
20-75-10-3	4323	66	1.810	240	0.471	377,653	46	3.127	397,163	0.534
20-75-10-4	3900	80	1.548	1	0.597	278	52	2.799	1,195,069	0.457

10,000 attempts, then we exclude that combination from our experiment. Out of 16 possible combinations for x and y, we could generate feasible random instances for eight combinations shown in Tables 1 and 2 resulting in a total of 40 problem instances. For simplicity, we use the same combinations for x and y for our real-life networks to obtain 40 real-life instances (and here, we allow trivial instances).

5.2. Results and observations

The computational times, optimality gaps, and costs of the best solutions found for the randomly-generated and real-life network instances are presented in Tables 1 and 3, respectively. The total preprocessing times and the number of processed nodes for all algorithms as well as the number of lazy cuts applied in the LBC algorithm along with the values of $|\Psi^V_{\mathbf{w},\eta}|$ and $|\Phi^V_{\mathbf{b},\beta}|$ for each randomly-generated and real-life instance are all listed in Tables 2 and 4, respectively.

All algorithms have some preprocessing step, which is included into their total runtime. The preprocessing step of the BC approach requires generating $\Psi^V_{\mathbf{w},\eta}$ and $\Phi^V_{\mathbf{b},\beta}$. Generating $\Psi^V_{\mathbf{w},\eta}$ is the only step required in the preprocessing for LBC. Besides generating sets $\Psi^V_{\mathbf{w},\eta}$ and $\Phi^V_{\mathbf{b},\beta}$, the CBB algorithm needs to calculate $|\widehat{\Phi}(\{Q\})|/\mathcal{C}(Q)$ (used in upper bounds) for all $Q \in \Psi^V_{\mathbf{w},\eta}$.

Given the one hour time limit for all algorithms, if an experiment exceeds the time limit during its preprocessing step, its runtime is shown as "PTL", i.e., "Preprocessing Time Limit". Then, if the BC algorithm exceeds the time limit while building the full IP model, its runtime is shown as "MTL", i.e., "Modeling Time Limit". Finally, if an algorithm exceeds the time limit during the actual solution process, its runtime is simply shown as "TL", which stands for "Time Limit".

Next, we summarize our observations for the randomly-generated instances; see Tables 1 and 2. In particular, we note that as seen in Table 1, all three algorithms show to be useful depending on the combinations of x and y values.

Consider the first four combinations of instances in Table 1, namely, 20-10-10-z, 20-10-40-z, 20-10-50-z and 20-10-75-z, i.e., x is fixed at 10, while the value of y increases. According to Table 1, the problem instances are easy when both x and y are small; all algorithms can solve instances with $x = y = 10$ within one second. Reasonably small values of x and y imply small cardinalities for sets $\Psi^V_{\mathbf{w},\eta}$ and $\Phi^V_{\mathbf{b},\beta}$; see Table 2. Therefore, the resulting IP formulations have relatively small number of variables and constraints, and the LBC algorithm also employs a small number of lazy cuts.

Table 3
Results for the real-life network instances (denoted as *name* − *x* − *y* − *z*). The running times are in seconds (Time). The optimality gaps (Gap), and the costs of the best solutions found (Solution) are also reported for each solution approach. The smallest run time (or the optimality gap) for each instance is highlighted and underlined.

Instance	BC			LBC			CBB		
	Time (seconds)	Gap (%)	Solution	Time (seconds)	Gap (%)	Solution	Time (seconds)	Gap (%)	Solution
karate-20-10-10	0.141	0.000	0.161	0.328	0.000	0.161	**0.126**	0.000	0.161
football-20-10-10	0.208	0.000	0.000	0.629	0.000	0.000	**0.150**	0.000	0.000
chesapeake-20-10-10	**0.122**	0.000	0.460	0.158	0.000	0.460	0.130	0.000	0.460
dolphins-20-10-10	0.515	0.000	0.196	0.186	0.000	0.196	**0.156**	0.000	0.196
lesmis-20-10-10	**0.186**	0.000	0.594	0.581	0.000	0.594	0.187	0.000	0.594
karate-20-10-40	19.715	0.000	54.188	**2.522**	0.000	54.188	TL	0.707	59.897
football-20-10-40	12.108	0.000	11.146	**0.584**	0.000	11.146	TL	0.783	11.162
chesapeake-20-10-40	10.638	0.000	31.819	**1.259**	0.000	31.819	TL	0.729	32.699
dolphins-20-10-40	22.520	0.000	14.814	**1.354**	0.000	14.814	TL	0.718	14.902
lesmis-20-10-40	37.817	0.000	7.381	**1.409**	0.000	7.381	TL	0.826	7.569
karate-20-10-50	451.886	0.000	40.998	**19.376**	0.000	40.998	TL	0.806	45.209
football-20-10-50	17.356	0.000	61.963	**2.796**	0.000	61.963	TL	0.882	70.770
chesapeake-20-10-50	206.996	0.000	103.317	**51.021**	0.000	103.317	TL	0.717	108.599
dolphins-20-10-50	607.392	0.000	40.081	**41.126**	0.000	40.081	TL	0.863	47.049
lesmis-20-10-50	619.729	0.000	55.288	**52.518**	0.000	55.288	TL	0.816	64.448
karate-20-10-75	**18.032**	0.000	263.120	26.248	0.000	263.120	TL	0.882	308.340
football-20-10-75	**15.588**	0.000	227.468	21.198	0.000	227.468	TL	0.945	271.613
chesapeake-20-10-75	8.510	0.000	276.955	**3.480**	0.000	276.955	TL	0.898	329.880
dolphins-20-10-75	**278.350**	0.000	439.314	TL	0.016	439.314	TL	0.876	483.881
lesmis-20-10-75	8.291	0.000	484.059	**3.323**	0.000	484.059	TL	0.906	528.032
karate-20-40-10	**390.964**	0.000	97.158	2419.990	0.000	97.158	TL	0.365	99.833
football-20-40-10	TL	0.577	126.478	TL	0.765	126.478	TL	**0.576**	151.936
chesapeake-20-40-10	805.997	0.000	83.917	**692.903**	0.000	83.917	TL	0.348	95.967
dolphins-20-40-10	1149.570	0.000	100.483	**145.066**	0.000	100.483	TL	0.477	127.196
lesmis-20-40-10	TL	**0.280**	121.566	TL	0.984	121.566	TL	0.426	121.566
karate-20-40-40	**1229.470**	0.000	627.145	1448.570	0.000	627.145	TL	0.815	899.999
football-20-40-40	TL	**0.434**	701.788	TL	0.588	899.373	TL	0.753	899.373
chesapeake-20-40-40	TL	0.853	901.258	TL	**0.697**	901.258	TL	0.827	901.258
dolphins-20-40-40	TL	**0.257**	791.308	TL	0.572	917.453	TL	0.760	917.453
lesmis-20-40-40	TL	0.246	906.370	TL	**0.221**	927.728	TL	0.758	927.728
karate-20-50-10	TL	0.924	295.292	TL	0.950	410.849	TL	**0.571**	326.787
football-20-50-10	TL	0.896	189.496	TL	0.964	189.496	TL	**0.686**	535.444
chesapeake-20-50-10	TL	0.972	337.353	TL	0.968	355.213	TL	**0.508**	294.376
dolphins-20-50-10	TL	0.820	179.735	TL	1.000	927.888	TL	**0.407**	714.307
lesmis-20-50-10	TL	0.546	235.287	TL	0.981	462.734	TL	**0.483**	235.287
karate-20-75-10	**8.855**	0.000	681.074	22.749	0.000	681.074	TL	0.343	686.118
football-20-75-10	**10.679**	0.000	575.241	264.111	0.000	575.241	TL	0.395	642.356
chesapeake-20-75-10	**4.760**	0.000	806.429	26.645	0.000	806.429	TL	0.190	806.429
dolphins-20-75-10	**8.552**	0.000	835.018	24.679	0.000	835.018	TL	0.335	845.634
lesmis-20-75-10	**7.160**	0.000	495.269	127.649	0.000	495.269	TL	0.240	495.269

For the CBB algorithm, a smaller size $\Psi^V_{\mathbf{w},\eta}$ implies a smaller search space, and if the size of $\Phi^V_{\mathbf{b},\beta}$ is also small, then the search tree is relatively shallow. Therefore, the CBB algorithm's performance is comparable with the other methods on instances with $x = y = 10$.

When we increase the value of y to 40 and 50, the value of β is about half of the total blocking costs of all vertices. Then, $|\Psi^V_{\mathbf{w},\eta}|$ remains relatively small but the size of $\Phi^V_{\mathbf{b},\beta}$ is considerably large as shown in Table 2. The LBC algorithm significantly outperforms the other two methods for these instances (i.e., 20-10-40-*z* and 20-10-50-*z*). Here, the IP formulations have relatively small number of variables, but very large number of constraints; the latter negatively affects the performance of the BC algorithm. As for the CBB algorithm, the search tree becomes deeper, and the feasibility checks take more time. Unlike the other two algorithms, these instances form a favorable scenario for the LBC algorithm as it deals with smaller IP models; see the number of lazy cuts in Table 2. Consequently, for these instances, the LBC algorithm is much faster than the other two algorithms.

If we further increase the value of y, the size of $\Phi^V_{\mathbf{b},\beta}$ decreases. We report instances with $y = 75$, since 75 is the maximum value of y for which we can randomly generate sufficient number of feasible instances. For instances 20-10-75-*z*, both sets $\Psi^V_{\mathbf{w},\eta}$ and $\Phi^V_{\mathbf{b},\beta}$ are relatively small, but set $\Phi^V_{\mathbf{b},\beta}$ is not as small as the case of $y = 10$ as shown in Table 2. These instances are still hard to solve for the CBB algorithm, but easy to solve for the BC and the LBC algorithms.

Next, we discuss instances, where we fix y at 10, but increase x, i.e., 20-10-10-*z*, 20-40-10-*z*, 20-50-10-*z* and 20-75-10-*z*. Instances are hard to solve when $|\Phi^V_{\mathbf{b},\beta}|$ is small but $|\Psi^V_{\mathbf{w},\eta}|$ remains large, i.e., 20-40-10-*z* and 20-50-10-*z* instances. For the majority of these instances, the search spaces are large, and the optimal solutions cannot be found within the time limit. However, the gaps returned by the CBB algorithm are consistently around 50% for both 20-40-10-*z* and 20-50-10-*z* instances. On the other hand, the gaps returned by the BC and LBC algorithms are not as consistent. This observation implies that for these instance classes, our combinatorial scheme provides better quality bounds than the linear programming relaxations in the BC and the LBC algorithms.

As we increase the value of x to 75 (i.e., 20-75-10-*z* instances), the size of $\Psi^V_{\mathbf{w},\eta}$ drops as shown in Table 2. Similar to the 20-10-75-*z* instances, the BC algorithm performs better than the other two algorithms. It specifically performs better than the LBC algorithm because finding constraints in a lazy fashion takes more time than finding all the blockers and solving a complete IP formulation.

The last combination to discuss is 20-40-40-*z*. For these instances both sets $\Psi^V_{\mathbf{w},\eta}$ and $\Phi^V_{\mathbf{b},\beta}$ are very large; see Table 2. Although none of the methods returns an optimal solution, the LBC algorithm always returns a nontrivial gap.

Table 4
Results for the real-life network instances (denoted as $name - x - y - z$). The numbers of η-minimal-clique-candidates ($|\Psi^V_{\mathbf{w},\eta}|$) and β-maximal-blockers ($|\Phi^V_{\mathbf{b},\beta}|$) for each instance are reported. The preprocessing times are in seconds (P_T). We also report the number of the search tree nodes (# node) for each algorithm, the number of lazy cuts (# cut) applied by the LBC algorithm, and the initial gap (Gap^0 (%)) of the CBB algorithm.

Instance	$\lvert\Psi^V_{\mathbf{w},\eta}\rvert$	$\lvert\Phi^V_{\mathbf{b},\beta}\rvert$	BC		LBC			CBB		
			P_T (seconds)	# node	P_T (seconds)	# node	# cut	P_T (seconds)	# node	Gap^0 (%)
karate-20-10-10	256	82	0.063	1	0.012	1	8	0.073	3	0.000
football-20-10-10	294	128	0.106	0	0.012	0	6	0.113	3	0.000
chesapeake-20-10-10	222	76	0.053	1	0.010	1	9	0.062	15	0.213
dolphins-20-10-10	279	97	0.081	1	0.012	1	8	0.093	3	0.000
lesmis-20-10-10	252	88	0.072	1	0.011	1	9	0.075	15	0.492
karate-20-10-40	184	23,009	9.369	22	0.010	559	98	9.416	969,385	0.967
football-20-10-40	210	14,901	7.299	1	0.008	73	107	8.047	119,997	0.930
chesapeake-20-10-40	181	12,999	5.988	51	0.008	774	102	6.829	475,881	0.968
dolphins-20-10-40	200	28,859	12.930	1	0.010	79	110	13.917	159,945	0.932
lesmis-20-10-40	250	32,219	17.056	1	0.011	289	228	17.437	34,717	0.890
karate-20-10-50	287	74,890	45.634	625	0.012	1676	475	45.942	297,341	0.983
football-20-10-50	209	19,343	10.356	55	0.009	668	178	11.000	1,424,957	0.981
chesapeake-20-10-50	483	33,616	34.629	1731	0.023	2240	212	35.489	193,219	0.941
dolphins-20-10-50	328	29,675	21.037	1313	0.012	5172	500	22.236	426,257	0.987
lesmis-20-10-50	425	21,210	19.074	1757	0.019	6009	347	19.558	484,491	0.980
karate-20-10-75	305	3471	7.203	1569	0.012	6064	451	7.962	3,047,705	0.949
football-20-10-75	230	6394	7.849	367	0.008	4060	487	8.667	2,060,613	0.983
chesapeake-20-10-75	249	4213	7.215	1	0.010	864	292	7.909	4,420,719	0.951
dolphins-20-10-75	321	4723	7.908	17,633	0.012	1,299,980	994	8.638	6,374,293	0.939
lesmis-20-10-75	217	4454	7.153	1	0.009	598	333	7.813	4,797,091	0.946
karate-20-40-10	15,416	193	13.470	17,271	1.196	18,802	15	14.075	38,575	0.660
football-20-40-10	12,325	169	8.670	61,952	0.922	1545	30	9.765	13,063	0.814
chesapeake-20-40-10	24,128	104	10.094	1319	1.649	1544	27	12.512	2575	0.625
dolphins-20-40-10	18,962	100	7.599	9270	1.385	42,412	16	9.343	45,917	0.746
lesmis-20-40-10	47,033	84	12.539	576,413	2.618	1879	28	17.774	25,371	0.845
karate-20-40-40	40,871	11,331	932.309	1	2.453	13,122	1831	1006.449	9	0.815
football-20-40-40	39,141	23,688	2036.320	1	2.234	2587	542	2105.410	5	0.753
chesapeake-20-40-40	28,654	37,749	2390.050	0	1.844	575	150	2402.590	3	0.827
dolphins-20-40-40	23,727	27,315	1363.070	1	1.773	716	269	1426.040	5	0.760
lesmis-20-40-40	36,314	28,273	2229.720	1	2.133	7631	1737	2361.730	3	0.758
karate-20-50-10	35,887	158	22.026	512	2.703	972	66	25.062	5703	0.754
football-20-50-10	26,620	138	14.582	10,865	2.904	2561	31	23.251	202,175	0.796
chesapeake-20-50-10	30,055	166	18.454	512	2.72	1272	75	27.436	8801	0.704
dolphins-20-50-10	46,063	107	22.187	2728	3.466	2552	29	25.863	2,555,371	0.544
lesmis-20-50-10	24,832	147	15.663	3925	2.304	1306	50	18.688	31,655	0.768
karate-20-75-10	3939	108	2.437	1	0.573	383	64	3.523	94,087	0.474
football-20-75-10	2490	101	1.677	1	0.470	606	87	2.348	98,665	0.579
chesapeake-20-75-10	6171	70	1.803	1	0.618	309	42	3.514	1,964,167	0.267
dolphins-20-75-10	4530	83	2.594	1	0.733	62	62	3.679	3,545,533	0.361
lesmis-20-75-10	2027	107	1.188	1	0.329	547	44	1.647	118,311	0.527

As we observe in Tables 3 and 4, the performances of all three algorithms with real-life networks are similar to those with the randomly-generated instances. This observation validates our analysis of their performances with the randomly-generated graphs, and further demonstrates the effects of the cardinalities of sets $\Phi^V_{\mathbf{b},\beta}$ and $\Psi^V_{\mathbf{w},\eta}$ on the running times and optimality gaps returned by the algorithms. It is important to note that for the real-life instances, none of the algorithms encounters either a preprocessing time limit or a modeling time limit. This might be due to the fact that the edge sets E in real-life instances are not empty, resulting in smaller sets $\check{E}$ and smaller search spaces. Finally, in Fig. 2(a) and (b), we illustrate optimal solutions for the CBRND problem in a randomly-generated instance and a real-life network, respectively; for the former recall that we use instances with $E = \emptyset$.

6. Conclusion

In this study, we consider the problem of constructing networks that are resilient to clique blockers. That is, we introduce the clique-blocker-resilient network design (CBRND) problem. We address the computational complexity of this problem, and explore its structural properties, which are then exploited to develop exact solution approaches including an integer programming (IP) formulation, a lazy-fashioned branch-and-cut (LBC) algorithm, and a combinatorial branch-and-bound (CBB) algorithm.

We report the results of our numerical experiments with randomly-generated and real-life networks to compare the performance of the proposed methods. Each method has its own advantages when solving instances with specific structures; this observation may provide a point of reference when selecting a specific solution method in various application settings. The IP model with an off-the-shelf solver may be a good choice when the size of the problem is relatively small. The CBB approach can provide the best optimality gaps when we have a very large collection of η-minimal-clique-candidates but a reasonably small collection of β-maximal-blockers. When we have an instance with the opposite situation, then the LBC algorithm becomes the best choice. The LBC algorithm is also the best approach to return nontrivial optimality gaps in challenging network instances.

Naturally, the proposed methods have some limitations; however, our results provide a number of interesting directions for future research. For example, improving bounding approaches in the combinatorial branch-and-bound algorithm could result in substantial running time improvements. Finally, it could be of interest to explore the problem of designing resilient networks to preserve other functional structures, e.g., the decision-maker could be inter-

ested in preserving large quasi-cliques (instead of cliques), or other clique relaxations.

Acknowledgments

Oleg Prokopyev was partially supported by National Science Foundation grant CBET-1803527. The authors would like to thank the anonymous referees for their constructive and helpful comments.

Appendix A. Discussion on the complexity of Algorithm 3

To find $\widehat{\Psi}(\emptyset)$ at the start of Algorithm 3, one needs to find $\widehat{\Phi}(\emptyset)$. That is, we need to verify for every $D \in \Phi^V_{\mathbf{b},\beta}$ whether there exists a $Q \in \Psi^V_{\mathbf{w},\eta}$ such that $Q \cap D = \emptyset$ and Q is a clique in G. Verifying these two conditions for a given D and Q can be done in $O(\eta\beta + \eta^2)$, and the whole procedure runs in $O((\eta\beta + \eta^2)n^{k\beta-1}n^{k\eta+1}) = O(\eta(\beta+\eta)n^{k(\eta+\beta)})$; recall the definition of k in Section 3.2 in the discussion of formulation (2a)-(2c). In particular, if all problem parameters are integers, then $k = 1$.

After finding $\widehat{\Phi}(\emptyset)$, to form $\widehat{\Psi}(\emptyset)$, we need to verify for each $Q \in \Psi^V_{\mathbf{w},\eta}$, whether there exists a $D \in \widehat{\Phi}(\emptyset)$ such that $Q \cap D = \emptyset$. This process runs in $O((\eta\beta)n^{k\beta-1}n^{k\eta+1}) = O((\eta\beta)n^{k(\eta+\beta)})$. Hence, computing $\widehat{\Psi}(\emptyset)$ can be done in $O(\eta(\beta+\eta)n^{k(\eta+\beta)})$.

The values of $|\Phi^V_{\mathbf{b},\beta} \setminus \widehat{\Phi}(\{Q\})|/\mathcal{C}(E(\{Q\}))$ for all $Q \in \Psi^V_{\mathbf{w},\eta}$ are also computed and sorted at the start of Algorithm 3 before the while loop. To compute $\widehat{\Phi}(\{Q\})$ for a given $Q \in \Psi^V_{\mathbf{w},\eta}$, we need to select every $D \in \widehat{\Phi}(\emptyset)$ (note that $\widehat{\Phi}(\emptyset)$ is computed when forming $\widehat{\Psi}(\emptyset)$ above) and check whether $Q \cap D = \emptyset$. Hence, $\widehat{\Phi}(\{Q\})$ for a given Q can be found in $O(\eta\beta n^{k\beta-1})$. Given that finding $\mathcal{C}(E(\{Q\}))$ takes $O(\eta^2)$, then the ratio $|\Phi^V_{\mathbf{b},\beta} \setminus \widehat{\Phi}(\{Q\})|/\mathcal{C}(E(\{Q\}))$ is computed in $O(\eta\beta n^{k\beta-1} + \eta^2)$ for each Q, and in

$$O\left(\eta\beta n^{k(\eta+\beta)} + \eta^2 n^{k\eta+1}\right) = O\left(\eta(\beta+\eta)n^{k(\eta+\beta)}\right)$$

for all Q. Sorting these values also takes time bounded by $O(n^{k\eta+1}\log(n^{k\eta+1}))$. To summarize, the time taken before the while loop in Algorithm 3 is $O(\eta(\beta+\eta)n^{k(\eta+\beta)} + n^{k\eta+1}\log(n^{k\eta+1}))$.

For each iteration of the while loop, sets $\widehat{\Phi}(S^*)$ and $\widehat{\Psi}(S^*)$ are formed by updating the corresponding sets found in the previous iteration. Updating set $\widehat{\Phi}(S^*)$ implies verifying whether each element of this set overlaps with a chosen η-minimal-clique-candidate, which runs in $O(\eta\beta n^{k\beta-1})$. Updating set $\widehat{\Psi}(S^*)$ can be done by examining each element of this set and verifying whether it does not have an overlap with some element of the updated set $\widehat{\Phi}(S^*)$. This procedure runs in $O(\eta\beta n^{k(\eta+\beta)})$. Thus, each iteration of the while loop runs in $O(\eta\beta n^{k\beta-1} + \eta\beta n^{k(\eta+\beta)}) = O(\eta\beta n^{k(\eta+\beta)})$. The number of iterations of the while loop is $O(n^{k\eta+1})$. Therefore, the while loop runs in $O(\eta\beta n^{k(\eta+\beta)+k\eta+1})$.

Appendix B. Proof of Proposition 3

Proposition 3 . *Given a tree node t^q, if $\hat{E} \subseteq \check{E}$ is a feasible solution to the CBRND problem that is within the subtree rooted at t^q, then*

$$\mathcal{C}(\hat{E}) \geq \mathcal{C}(E(S_q)) + \max\left\{\delta(D) \;:\; D \in \widehat{\Phi}(S_q)\right\},$$

where

$$\delta(D) = \min\{\Delta\mathcal{C}_q(Q) \;:\; Q \in T_q,\; Q \cap D = \emptyset\}.$$

Proof. Since $\hat{E}$ is located within the subtree rooted at node t^q, then $E(S_q)$ is part of $\hat{E}$ and

$$\mathcal{C}(\hat{E}) = \mathcal{C}(E(S_q)) + \mathcal{C}\left(\hat{E} \setminus E(S_q)\right). \tag{B.1}$$

Consider an arbitrary β-maximal-blocker $D' \in \widehat{\Phi}(S_q)$. Since $\hat{E}$ is a feasible solution to the CBRND problem located within the subtree rooted at node t^q, by Proposition 1, there exists an η-minimal-clique-candidate $Q' \in T_q$ such that $Q' \cap D' = \emptyset$ and $(E(\{Q'\}) \setminus E(S_q)) \subseteq (\hat{E} \setminus E(S_q))$. Then,

$$\mathcal{C}\left(\hat{E} \setminus E(S_q)\right) \geq \mathcal{C}\left(E\left(\{Q'\}\right) \setminus E(S_q)\right) = \Delta\mathcal{C}_q(Q') \geq \delta(D'). \tag{B.2}$$

Using (B.1) and (B.2), we have

$$\mathcal{C}(\hat{E}) \geq \mathcal{C}(E(S_q)) + \delta(D').$$

Note that D' is an arbitrarily selected β-maximal-blocker in $\widehat{\Phi}(S_q)$. Hence, $\mathcal{C}(\hat{E}) \geq \mathcal{C}(E(S_q)) + \max\{\delta(D) \;:\; D \in \widehat{\Phi}(S_q)\}$. □

References

Afshari Rad, M., & Kakhki, H. T. (2017). Two extended formulations for cardinality maximum flow network interdiction problem. *Networks, 69*(4), 367–377.

Altner, D. S., Ergun, Ö., & Uhan, N. A. (2010). The maximum flow network interdiction problem: Valid inequalities, integrality gaps, and approximability. *Operations Research Letters, 38*(1), 33–38.

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

Bazgan, C., Toubaline, S., & Tuza, Z. (2011). The most vital nodes with respect to independent set and vertex cover. *Discrete Applied Mathematics, 159*(17), 1933–1946.

Bazgan, C., Toubaline, S., & Vanderpooten, D. (2013). Critical edges/nodes for the minimum spanning tree problem: Complexity and approximation. *Journal of Combinatorial Optimization, 26*(1), 178–189.

Chen, C.-L., Zheng, Q. P., Veremyev, A., Pasiliao, E. L., & Boginski, V. (2021). Failure mitigation and restoration in interdependent networks via mixed-integer optimization. *IEEE Transactions on Network Science and Engineering, 8*(2), 1293–1304. https://doi.org/10.1109/TNSE.2020.3005193.

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

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

Frederickson, G. N., & Solis-Oba, R. (1999). Increasing the weight of minimum spanning trees. *Journal of Algorithms, 33*(2), 244–266.

Furini, F., Ljubi, I., Martin, S., & San Segundo, P. (2019). The maximum clique interdiction problem. *European Journal of Operational Research, 277*(1), 112–127. https://doi.org/10.1016/j.ejor.2019.02.028.

Garey, M., & Johnson, D. (1979). *Computers and intractability: A guide to the theory of NP-completeness*. Freeman and Co., New York.

Ghare, P., Montgomery, D. C., & Turner, W. (1971). Optimal interdiction policy for a flow network. *Naval Research Logistics Quarterly, 18*(1), 37–45.

Haimes, Y. Y. (2009). On the definition of resilience in systems. *Risk Analysis: An International Journal, 29*(4), 498–501.

Holling, C. S. (1996). Engineering resilience versus ecological resilience. *Engineering within ecological constraints, 31*(1996), 32.

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

Khachiyan, L., Boros, E., Borys, K., Elbassioni, K., Gurvich, V., Rudolf, G., et al., (2008). On short paths interdiction problems: Total and node-wise limited interdiction. *Theory of Computing Systems, 43*(2), 204–233.

Lin, K.-C., & Chern, M.-S. (1993). The most vital edges in the minimum spanning tree problem. *Information Processing Letters, 45*(1), 25–31.

Mahdavi Pajouh, F. (2019). Minimum cost edge blocker clique problem. *Annals of Operations Research*, 1–32.

Mahdavi Pajouh, F., Boginski, V., & Pasiliao, E. L. (2014). Minimum vertex blocker clique problem. *Networks, 64*(1), 48–64.

Mahdavi Pajouh, F., Walteros, J. L., Boginski, V., & Pasiliao, E. L. (2015). Minimum edge blocker dominating set problem. *European Journal of Operational Research, 247*(1), 16–26.

Nasirian, F., Mahdavi Pajouh, F., & Namayanja, J. (2019). Exact algorithms for the minimum cost vertex blocker clique problem. *Computers and Operations Research, 103*, 296–309.

Pardalos, P. M., & Xue, J. (1994). The maximum clique problem. *Journal of Global Optimization, 4*(3), 301–328. https://doi.org/10.1007/BF01098364.

Ries, B., Bentz, C., Picouleau, C., de Werra, D., Costa, M.-C., & Zenklusen, R. (2010). Blockers and transversals in some subclasses of bipartite graphs: When caterpillars are dancing on a grid. *Discrete Mathematics, 310*(1), 132–146.

Rutenburg, V. (1994). Propositional truth maintenance systems: Classification and complexity analysis. *Annals of Mathematics and Artificial Intelligence, 10*(3), 207–231.

Ryan, A. R., & Nesreen, K. A. (2015). The network data repository with interactive graph analytics and visualization. In *Proceedings of the twenty-ninth AAAI conference on artificial intelligence*. http://networkrepository.com

Schieber, B., Bar-Noy, A., & Khuller, S. (1995). The complexity of finding most vital arcs and nodes.

Sharkey, T. C., Nurre Pinkley, S. G., Eisenberg, D. A., & Alderson, D. L. (2021). In search of network resilience: An optimization-based view. *Networks, 77*(2), 225–254.

Shen, Y., Nguyen, N. P., Xuan, Y., & Thai, M. T. (2012). On the discovery of critical links and nodes for assessing network vulnerability. *IEEE/ACM Transactions on Networking, 21*(3), 963–973.

Tang, Y., Richard, J.-P. P., & Smith, J. C. (2016). A class of algorithms for mixed-integer bilevel min–max optimization. *Journal of Global Optimization, 66*(2), 225–262.

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

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

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

Veremyev, A., Prokopyev, O. A., & Pasiliao, E. L. (2019). Finding critical links for closeness centrality. *INFORMS Journal on Computing, 31*(2), 367–389.

Wei, N., Walteros, J. L., & Pajouh, F. M. (2021). Integer programming formulations for minimum spanning tree interdiction. *INFORMS Journal on Computing, 33*(4), 1461–1480. https://doi.org/10.1287/ijoc.2020.1018. https://pubsonline.informs.org/doi/abs/10.1287/ijoc.2020.1018

Wollmer, R. (1964). Removing arcs from a network. *Operations Research, 12*(6), 934–940.

Wong, P., Sun, C., Lo, E., Yiu, M. L., Wu, X., Zhao, Z., et al., (2017). Finding k most influential edges on flow graphs. *Information Systems, 65*, 93–105.

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

Wu, Q., & Hao, J.-K. (2015). A review on algorithms for maximum clique problems. *European Journal of Operational Research, 242*(3), 693–709.

Yodo, N., & Wang, P. (2016). Engineering resilience quantification and system design implications: A literature survey. *Journal of Mechanical Design, 138*(11).

Zenklusen, R. (2010a). Matching interdiction. *Discrete Applied Mathematics, 158*(15), 1676–1690.

Zenklusen, R. (2010b). Network flow interdiction on planar graphs. *Discrete Applied Mathematics, 158*(13), 1441–1455.