A Reduction-based Algorithm for the Clique Interdiction Problem

Chenghao Zhu , **Yi Zhou**[*] , **Haoyu Jiang**
University of Electronic Science and Technology of China
axs7384@gmail.com, zhou.yi@uestc.edu.cn, FirstSSAT@outlook.com,

Abstract

The Clique Interdiction Problem (CIP) aims to minimize the size of the largest clique in a given graph by removing a given number of vertices. The CIP models a special Stackelberg game and has important applications in fields such as pandemic control and terrorist identification. However, the CIP is a bilevel graph optimization problem, making it very challenging to solve. Recently, data reduction techniques have been successfully applied in many (single-level) graph optimization problems like the vertex cover problem. Motivated by this, we investigate a set of novel reduction rules and design a reduction-based algorithm, RECIP, for practically solving the CIP. RECIP enjoys an effective preprocessing procedure that systematically reduces the input graph, making the problem much easier to solve. Extensive experiments on 124 large real-world networks demonstrate the superior performance of RECIP and validate the effectiveness of the proposed reduction rules.

1 Introduction

1.1 Problem Background

The maximum clique problem (MCP) is fundamental and well-studied in graph theory and combinatorial optimization. The size of the maximum clique is an important metric to measure the density or cohesion of graphs. A larger size of the maximum clique implies a denser graph structure [Borgatti *et al.*, 2024], often corresponding to tightly connected communities in applications such as social networks, biological networks, and signal processing [Van Cleemput, 2012; Dunbar and Spoors, 1995; Malod-Dognin *et al.*, 2010; Douik *et al.*, 2020].

In many real-world applications, it is important not only to identify large cliques but also to disrupt or minimize their sizes through a process known as interdiction [Dempe, 2020]. This leads to the notion **Clique Interdiction Problem (CIP)**. The CIP is defined as follows: Given a graph G and an interdiction budget value k, decide at most k vertices to be removed from (, or interdicted in) the graph such that the size of the maximum clique in the remaining graph is minimized. From the lens of application, the problem is used to identify critical nodes in various real-world networks. One can see that the vertices for interdiction are typically among the most important or influential in the graph, as their removal has the greatest impact on the graph structure. We list some specific applications in the following.

[*]Corresponding author.

Pandemic Control For epidemic control, identifying important nodes for disease spreading is a central topic in network epidemiology, as large cliques play a significant role in the spread of diseases [Wang *et al.*, 2012; Šikić *et al.*, 2013]. Therefore, it is essential to identify critical nodes that can reduce the size of cliques and monitor these nodes to control the spread of epidemics [Valdez *et al.*, 2023; Grass and Fischer, 2016].

Terrorist Identification Large cliques can be potential sources of catastrophic events, such as terrorist attacks or cyberattacks [Sageman, 2004; Berry *et al.*, 2004]. Cliques promote cohesion and solidarity, enabling large groups within terrorist or criminal networks to coordinate devastating actions. Therefore, monitoring and regulating the cohesiveness of terrorist networks is of critical importance.

1.2 Related Literature

The general problem of interdicting several vertices or edges such that the graph becomes less cohesive is receiving increasing attention. Specific formulations include the clique interdiction problem in the paper, the minimum vertex blocker clique problem, which minimizes the number of vertices removed to ensure the maximum clique in the remaining graph is below a specified size [Nasirian *et al.*, 2019; Mahdavi Pajouh *et al.*, 2014], and the edge interdiction clique problem, which minimizes the maximum clique size after removing at most k edges [Furini *et al.*, 2021; Mattia, 2024]. In the optimization community, such problems are classified as bilevel optimization problems and two-stage stochastic optimization problems with recourse(2SPRs) [Dempe, 2020]. In particular, the CIP models a special Stackelberg game where the leader interdicts a set of vertices and the follower maximizes the clique in the remaining graph [Xiao *et al.*, 2014]. In this sense, the problem is different from another recently studied problem called the k-defective clique problem, which maximizes the maximum clique size after adding k edges

[Luo *et al.*, 2024; Chen *et al.*, 2021] because adding edges and maximizing the clique are not adversarial.

From the perspective of computational intractability, the CIP is challenging as the decision of maximum clique size in the remaining graph is NP-hard [Karp, 2010], W[1]-hard [Downey and Fellows, 2012], and hard to approximate [Hastad, 1996]. In fact, the decision version of CIP has been shown to be Σ_2^p-complete [Rutenburg, 1994; Grüne and Wulf, 2024]. Nevertheless, there are at least two existing algorithms for solving the CIP practically and optimally. One method is to formulate the problem as a bilevel integer linear program (BILP), and then rely on the BILP solver to obtain the solution [Becker, 2017; Tang *et al.*, 2016; Fischetti *et al.*, 2017]. However, this approach may be inefficient because it does not make full use of the specific structure of the CIP. Another algorithm called CLINTER-INTER, probably the state-of-the-art algorithm for the CIP to our knowledge, recasts the problem to a normal single-level integer linear problem with an exponential number of rows [Furini *et al.*, 2019], then uses the maximum clique solver for constraint generation to solve this ILP. Clearly, the scale of the linear program depends on the input graph. Therefore, when the input graph is huge, which is often the case in real-world scenarios, efficient reduction pre-processing techniques can be useful for reducing the size of the linear program. This in turn improves the efficiency of the final algorithm. On the other hand, the data reduction technique has been widely used for simplifying NP-hard problems, like vertex cover [Hespe *et al.*, 2020], independent set [Xiao *et al.*, 2021], or cluster editing [Bläsius *et al.*, 2022]. A nice recent review in [Abu-Khzam *et al.*, 2022] pointed out the possibility of extensive application of this technique for harder discrete problems. The study of reduction rules and reduction-based algorithms for interdiction optimization problems like the basic CIP is still limited in existing literature. Motivated by this gap, we propose an efficient algorithm for the CIP with a particular focus on data reduction, enabling better scalability and efficiency.

1.3 Our Contributions

Our contributions are mainly two-fold.

First, we investigate the data reduction techniques for the CIP. We propose novel reduction rules including *color*, *exact clique*, *triangle*, *interdiction*, and the *domination* reduction rules, which are used to simplify the input. These rules rely on an in-depth structural analysis and can reduce the input, i.e., the graph G and the budget k, with an optimality guarantee.

We secondly provide a reduction-based algorithm, RECIP, for solving the CIP in real-world graphs. Based on the reduction rules, we provide a unified reduction algorithm that preprocesses the input instance. Worst-case time complexity guarantees for each reduction step are also given. We also provide polynomial-time algorithms for finding lower bounds that are tighter than current methods for some instances. Finally, the RECIP integrates the reduction algorithm and the lower bound estimations into a branch-and-cut framework.

Extensive experiments demonstrate that RECIP outperforms the state-of-the-art methods on nearly 90% of real-world networks. Notably, RECIP features a powerful graph reduction ability in preprocessing. A reduction of 85% vertices is often observed for most graphs. Some reduction steps which are not polynomial-time in the worst case (due to the invocation of the maximum clique oracle) perform still efficiently in practice. The source codes are publicly available [1].

2 Preliminary

Let $G = (V, E)$ be a simple finite undirected graph, where V is the vertex set and E is the edge set of G. When the context is clear, we use n to denote the number of vertices $|V|$ and m to denote the number of edges $|E|$ of the graph. The *open neighborhood* of a vertex $v \in V$ is the set of its adjacent vertices, $N_G(v) = \{u \mid \{u, v\} \in E\}$, and the size of the set is the degree of v, $d(v) = |N_G(v)|$. We further define the *closed neighborhood* of a vertex $v \in V$ to be $N_G[v] = N(v) \cup \{v\}$. For convenience, we use $N(u)$ to denote $N_G(u)$ and $N[u]$ to denote $N_G[v]$ unless otherwise specified.

A vertex set $C \subseteq V$ is called a *clique* if every pair of distinct vertices $u, v \in C$ satisfies $\{u, v\} \in E$. The maximum clique size of G, which is the size of the largest clique in G, is denoted by $\omega(G)$. We denote $T_c(n)$ as the time complexity of computing $\omega(G)$, where n is the number of vertices in G. The current computation of $\omega(G)$ is in fact very fast in large, sparse graphs empirically even though the best-known $T_c(n)$ is still exponential $O^*(1.2^n)$ [Xiao and Nagamochi, 2017] in theory. In the paper, we employ the well-performed algorithm in [Chang, 2019] to compute $\omega(G)$ and find the maximum clique in the graphs. Furthermore, a clique is referred to as a *maximal clique* if it is not a proper subset of any other clique. Given a vertex set $S \subseteq V$, the subgraph of G induced by S is denoted by $G[S]$.

Given a graph $G = (V, E)$ and a nonnegative integer k, $\theta(G, k)$ represents the minimum value of the size of the maximum clique in the graph obtained by removing at most k vertices from G. Formally,

$$\theta(G, k) = \min_{S \subseteq V, |S| \leq k} \omega(G[V \setminus S]).$$

The set $\mathcal{S}(G, k)$ refers to all subsets S that obtain the optimal $\theta(G, k)$, i.e. $\mathcal{S}(G, k) = \{S | S = \arg\max_{S \subseteq V, |S| \leq k} \omega(G[V \setminus S])\}$, for a given G and k.

3 Reduction Rules

In the paper, the reduction rule refers to the rule that reduces the size of the problem instance, for example, the number of vertices and edges of graph G, or the budget value k, while preserving optimality. Before introducing these reduction rules, we assume that a lower bound value lb of $\theta(G, k)$ is known beforehand. We defer the methods of obtaining effective lower bounds to Section 4.1.

Exact Clique Reduction Our first reduction is based on a simple observation: If the size of the maximum clique containing vertex u is less than $\theta(G, k)$, then removing vertex u does not affect the solution to this instance G and k. As

[1] https://github.com/axs7385/RECIP

$\theta(G,k)$ is not known beforehand, we use the lower bound of lb instead in the following reduction rule.

Lemma 1 (Exact Clique Reduction)**.** *Given a graph $G = (V,E)$, an integer k, and an integer lb, if $\theta(G,k) \geq lb$ and there is a vertex $u \in V$ such that $\omega(G[N(u)]) \leq lb - 2$, then $\theta(G,k) = \theta(G[V \setminus \{u\}])$.*

Proof of this lemma, as well as missing proofs in the remainder of the paper, are left in the appended file.

Suppose that $\theta(G,k) \geq lb$. We can exhaustively remove all vertices $u \in V$ that $\omega(G[N(u)]) \leq lb - 2$ based on this reduction rule. The time complexity of this reduction is $O\left(\sum_{u\in V} T_c(d(u))\right)$. Specifically, we observe that the removal of a vertex from the graph does not make other vertices satisfy this removable condition by Lemma 2. So we only need to compute the maximum clique size in $G[N(u)]$ for each u.

Lemma 2. *Given a graph $G = (V,E)$, an integer k, and two adjacent vertices $u, v \in V$, if $\omega(G[N(u)]) < \omega(G[N(v)])$, then $\omega(G[N(v) \setminus \{u\}]) = \omega(G[N(v)])$.*

Degree Reduction For the efficiency consideration, we hope to avoid calculating the $\omega(G[N(u)])$ for all $u \in V$ if the input graph is huge. Hence, we propose the degree reduction rule, which is obtained from the fact that $d(u)$ is an upper bound for $\omega(G[N(u)])$.

Lemma 3 (Degree Reduction)**.** *Given a graph $G = (V,E)$, an integer k and an integer lb, if $\theta(G,k) \geq lb$ and there is a vertex $u \in V$ such that $d(u) \leq lb - 2$, then $\theta(G,k) = \theta(G[V \setminus \{u\}], k)$.*

The correctness of Lemma 3 can be directly derived from Lemma 1, using the fact that $\omega(G[N(u)]) \leq d(u)$.

The degree reduction was also used in CLIQUE-INTER [Furini *et al.*, 2019], the best-known existing solver for the CIP. Suppose that $\theta(G,k) \geq lb$, the time complexity of exhaustively removing all vertices $u \in V$ that $d(u) \leq lb - 2$ is linear time, $O(m)$. This is based on the observation that, when a vertex u is removed, the degree of vertices in $N(u)$ is decreased by 1, making these vertices removable as well. Similar to the k-core computation where a linear-heap is used to maintain the vertices of different degrees [Chang and Qin, 2019], one can obtain the linear time reduction procedure.

Color Reduction A coloring of a graph G is a mapping $c : V \to Col$ from the vertex set V to a color set Col (namely, a set of integers), such that no vertex shares the same color with any of its neighbors. Suppose that a coloring c is given for a graph G. For a vertex u, the number of distinct colors assigned to all vertices in $N(u)$, $|\{c(v) : v \in N(u)\}|$, is an upper bound on the size of the maximum clique in $G[N(u)]$. We denote this value by $ds_c(u)$ and refer to it as the *saturation* of a vertex u.

Lemma 4 (Color Reduction)**.** *Given a graph $G = (V,E)$, an integer k, an integer lb and a coloring c of G, if $\theta(G,k) \geq lb$ and there is a vertex $u \in V$ such that $ds_c(u) \leq lb - 2$, then $\theta(G,k) = \theta(G[V \setminus \{u\}])$.*

The correctness of Lemma 4 can also be directly derived from Lemma 1, using the fact that $\omega(G[N(u)]) \leq ds_c(u)$.

Given a coloring c of the graph G, suppose $\theta(G,k) \geq lb$. We can exhaustively remove all vertices $u \in V$ that $ds_c(u) \leq lb - 2$ based on this reduction rule. The time complexity of this operation can be simply achieved in linear time $O(m)$. In order to obtain an initial coloring, we use the heuristic coloring algorithm in the state-of-the-art maximum clique algorithm [Li *et al.*, 2017]. Indeed, we can remove more vertices if the colors of the vertices are dynamically adjusted during the reduction process. This results in a more complex reduction process, with a running time of $O(nm)$. The details are provided in the appended file.

Triangle Reduction For an edge $\{u,v\}$, if the number of common neighbors of u and v, $|N(u) \cap N(v)|$, is less than $\theta(G,k) - 2$, then the size of the clique containing both vertices is smaller than $\theta(G,k) - 2$. This implies the triangle reduction rule.

Lemma 5 (Triangle Reduction)**.** *Given a graph $G = (V,E)$, an integer k and an integer lb, if $\theta(G,k) \geq lb$ and there is an edge $\{u,v\} \in E$ such that $|N(u) \cap N(v)| \leq lb - 3$ then $\theta(G,k) = \theta((V, E \setminus \{\{u,v\}\}), k)$.*

Suppose that $\theta(G,k) \geq lb$. The triangle reduction rule indicates that we can exhaustively remove all edges $\{u,v\} \in E$ that $|N(u) \cap N(v)| \leq lb - 3$ based on this reduction rule. In order to identify such edges, we can list all triangles (3-cliques) in the G, which can be done in time $O(m^{1.5})$ [Latapy, 2008]. Then, the size of $|N(u) \cap N(v)|$ is equal to the number of triangles including edge $\{u,v\}$. Moreover, if $\{w,u,v\}$ forms a triangle, the removal of edge $\{u,v\}$ also makes edges $\{w,u\}$ or $\{w,v\}$ involved in fewer triangles. In other words, we need to keep updating the number of triangles that each edge is involved with. This can be also done in linear time $O(3m)$ because each edge can be removed at most once. In total, the running time of exhaustively removing all edges is $O(m^{1.5})$.

The relationship between triangle reduction and edges is analogous to that between degree reduction and vertices. Similarly, we could extend this concept to reductions based on color numbers or maximum cliques of common neighbors.

Lemma 6 (Triangle Clique Reduction)**.** *Given a graph $G = (V,E)$, an integer k and an integer lb, if $\theta(G,k) \geq lb$ and there is an edge $\{u,v\} \in E$ such that $\omega(G[N(u) \cap N(v)]) \leq \theta(G,k) - 3$, then $\theta(G,k) = \theta((V, E \setminus \{\{u,v\}\}), k)$.*

Lemma 7 (Triangle Color Reduction)**.** *Given a graph $G = (V,E)$, an integer k, and an integer lb, if $\theta(G,k) \geq lb$ and there is an edge $\{u,v\} \in E$ such that $|\{c(w) : w \in N(u) \cap N(v)\}| \leq \theta(G,k) - 3$, then $\theta(G,k) = \theta((V, E \setminus \{\{u,v\}\}), k)$.*

Clearly, exhaustively removing edges that satisfy the triangle clique or triangle color reduction rules is more computationally expensive than that of triangle reduction. On the other hand, after exhaustively applying the degree, color, and triangle reductions, the graph becomes significantly denser, making it hard to remove edges meeting the triangle clique and color reduction rule. Thus, the above two reductions are only used when a long computational time is allowed.

Interdiction Reduction The previous reductions primarily identify vertices that are not in $\mathcal{S}(G,k)$ or not in any maximum clique of G. In contrast, now we introduce interdiction reduction rules that identify vertices that must be a part of every maximum clique in the remaining graph.

Lemma 8 (Interdiction Reduction). *Given a graph $G = (V, E)$, an integer $k > 0$, and a vertex $u \in V$, if for any $v \in V \setminus N[u]$, $\omega(G[N(u)]) - k > \omega(G[N(v)])$, then $\theta(G, k) = \theta(G[V \setminus \{u\}], k - 1)$.*

In other words, if the largest clique containing vertex u is larger than any maximum clique that excludes u by more than k, then in any solution where at most k vertices are removed and u is not among them, the largest remaining clique must include u.

We can remove all vertices $u \in V$ that for any $v \in V \setminus N[u], \omega(G[N(u)]) - k > \omega(G[N(v)])$ based on this reduction rule. The time complexity of this reduction is $O\left(n^2\right)$ when we have already computed $\omega(G[N(u)])$ for all vertices $u \in V$ in the exact clique reduction above.

Domination Reduction If all the neighbors of a vertex v are also neighbors of another vertex u, and u has additional neighbors that are not connected to v, then removing vertex u is always more effective than removing vertex v. We call this property u dominates v.

Lemma 9 (Domination Reduction). *Given a graph $G = (V, E)$, an integer k, and two vertices $u, v \in V$, if $N(v) \subset N(u)$ or $N[v] \subset N[u]$, then if there exists a set $S \in \mathcal{S}(G, k)$ such that $v \in S$ and $u \notin S$, then $(S \setminus \{v\}) \cup \{u\} \in \mathcal{S}(G, k)$.*

We can find all pairs of vertices $u, v \in V$ that u dominates v based on this reduction rule. The time complexity of this reduction is $O\left(nm\right)$ because the time complexity of determining the domination relationship for a pair of vertices (u, v) is $O(d(u) + d(v))$.

4 The Reduction-and-Search Algorithm

4.1 Lower Bound Estimation

The proposed reduction rules rely on a known lower bound of $\theta(G, k)$. In this section, we elaborate on the algorithm to obtain an effective lower bound.

The Linear Program Relaxation Given a graph $G = (V, E)$ and an integer k, it is known that $\theta(G, k)$ can be computed using the following ILP formulation [Furini *et al.*, 2019].

$$\text{ILP-CIP}(G,k,\mathcal{C}) \begin{cases} \min \quad y \\ \quad \text{s.t.} \sum_{u \in V} x_u \geq n - k, \\ \quad \sum_{u \in C} x_u \leq y, \quad \forall C \in \mathcal{C}, \\ \quad y \in \mathbb{R}, x_u \in \{0, 1\}, \quad \forall u \in V \end{cases}$$

where $\mathcal{C}$ is the set of all cliques in the graph G. Note that the binary $x_u = 0$ means that vertex u is interdicted.

Clearly, we can obtain a lower bound for $\theta(G, k)$ by relaxing the ILP. A common relaxation technique is modifying the domain of x_u to all real values in the range $[0, 1]$. However, it is still difficult to obtain the optimal solution of this LP due to the exponential number of cliques in G. Yet it is hard to separate the inequality in polynomial time.

Due to the exponential size of $|\mathcal{C}|$, in the paper, we provide an alternative relaxation by restricting $\mathcal{C}$ to only a subset of the maximal cliques in the graph. We note that the following observation holds.

Lemma 10. *Assume $\mathcal{C} = \{C_1, ..., C_p\}$ where $C_1, ..., C_p$ are cliques in G (, probably not all cliques in G). Then the optimal solution to ILP-CIP$(G, k, \mathcal{C})$ is a lower bound of $\theta(G, k)$.*

Restrict $\mathcal{C}$ to Disjoint Cliques By Lemma 10, when $\mathcal{C}$ only includes some mutually disjoint cliques of G, the ILP-CIP$(G, k, \mathcal{C})$ is clearly a lower bound of $\theta(G, k)$. We refer to this lower bound as the *disjoint lower bound*.

To compute a disjoint lower bound, we first construct a set $\mathcal{C}$ which consists of disjoint cliques in G using the simple Algorithm 2 in the appended file. This algorithm uses the simple greedy strategy and has a time complexity of $O(n^2)$.

If $\mathcal{C}$ consists solely of disjoint cliques of G, then ILP-CIP$(G, k, \mathcal{C})$ can be computed without the need to invoke an ILP solver. Denote $f(\mathcal{C}, y)$ as the minimum number of vertices that need to be removed such that the size of the largest clique in $\mathcal{C}$ is less than or equal to y, i.e., $f(\mathcal{C}, y) = \sum_{C \in \mathcal{C}} \max(0, |C| - y)$. The smallest y such that $f(\mathcal{C}, y) \leq k$ is the result of ILP-CIP$(G, k, \mathcal{C})$. Because $f(\mathcal{C}, y)$ is non-decreasing as y increases, allowing us to use binary search to determine y, ILP-CIP$(G, k, \mathcal{C})$ is obtained in $O(|\mathcal{C}| \log n)$. In sum, the time to obtain a disjoint lower bound is $O(n^2 + |\mathcal{C}| \log n) = O(n^2)$ when $\mathcal{C}$ only contains disjoint cliques.

Relaxation to Bipartite Cliques Furthermore, supposing that $\mathcal{C}$ can be partitioned into $\mathcal{C}_1$ and $\mathcal{C}_2$, where $\mathcal{C}_1$ includes some mutually disjoint cliques of G, and $\mathcal{C}_2$ also includes some mutually disjoint cliques of G, then ILP-CIP$(G, k, \mathcal{C})$ is also a lower bound of $\theta(G, k)$ by Lemma 10. We refer to this lower bound as the *bipartite lower bound*.

In order to obtain $\mathcal{C}_1$ and $\mathcal{C}_2$, we simply run the greedy Algorithm 2 in the appended file twice. Then, we use a flow-based algorithm to compute ILP-CIP$(G, k, \mathcal{C} = \mathcal{C}_1 \cup \mathcal{C}_2)$.

For notational convenience, let us denote $f'(\mathcal{C} = \mathcal{C}_1 \cup \mathcal{C}_2, y)$ as the minimum number of vertices that need to be removed such that the size of the largest clique in $\mathcal{C}$ is less than or equal to y, i.e. $f'(\mathcal{C}, y) = \sum_{C \in \mathcal{C}_1} \max(0, |C| - y) + \sum_{C \in \mathcal{C}_2} \max(0, |C| - y) - h(\mathcal{C}_1, \mathcal{C}_2, y)$. Here, $h(\mathcal{C}_1, \mathcal{C}_2, y)$ denotes the maximum number of vertices that can be removed to simultaneously reduce the size of cliques in $\mathcal{C}_1$ and $\mathcal{C}_2$ to at least y.

Given sets $\mathcal{C}_1$ and $\mathcal{C}_2$, and an integer y, the $h(\mathcal{C}_1, \mathcal{C}_2, y)$ is computed using a maximum flow algorithm. We first build a flow network as shown in Figure 1. For each clique C in $\mathcal{C}_1$ and $\mathcal{C}_2$, we create a node in the network representing the C. We also create an additional source node S and a target node T. Each node representing a $C_1 \in \mathcal{C}_1$ connects to the source S with capacity $\max(0, |C_1| - y)$, and each node representing $C_2 \in \mathcal{C}_2$ connects to the target T with capacity $\max(0, |C_2| - y)$. Edges between $C_1 \in \mathcal{C}_1$ and $C_2 \in \mathcal{C}_2$ have capacity $|C_1 \cap C_2|$. The maximum (S, T)-flow in this flow network is equal to $h(\mathcal{C}_1, \mathcal{C}_2, y)$. The total time of building the flow network and computing the maximum flow is $O(n^3)$.

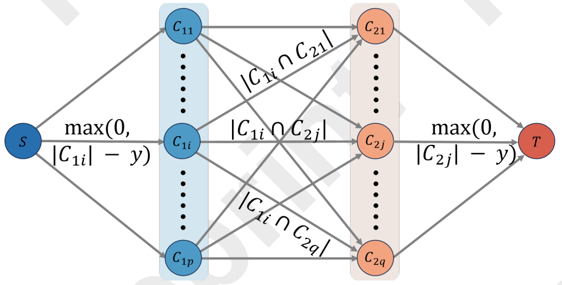


Figure 1: Structure of the network flow model, the value on the side represents the flow size.

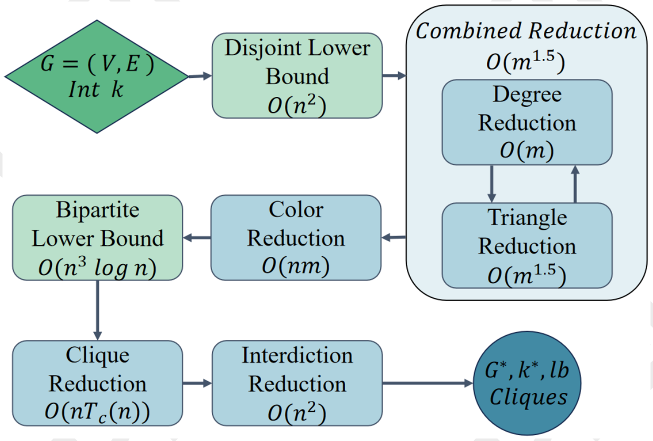
As $h(\mathcal{C}_1, \mathcal{C}_2, y)$ can be computed efficiently, it is easy to compute ILP-CIP$(G, k, \mathcal{C} = \mathcal{C}_1 \cup \mathcal{C}_2)$. Similar to the disjoint lower bound, the smallest y such that $f'(\mathcal{C} = \mathcal{C}_1 \cup \mathcal{C}_2, y) \leq k$ is the result of ILP-CIP$(G, k, \mathcal{C} = \mathcal{C}_1 \cup \mathcal{C}_2)$. Thereby, the total running time of computing the bipartite lower bound is $O(n^3 \log n)$.

4.2 The Reduction-based Preprocess

Firstly, these reduction rules require a valid lower bound of $\theta(G, k)$. Hence, the initial step is to use the simple disjoint cliques relaxation method to obtain a lower bound.

Afterwards, we call the degree reduction and triangle reduction to reduce the (G, k). The two reductions can be done in an intertwined manner. Specifically, the deletion of an edge decreases the degree of its connected vertices, while the deletion of a vertex affects the triangles that an edge involves. Hence, we maintain the vertex degrees and triangles simultaneously and update them when a vertex (and its incident edges) or an edge is removed from the graph. The total time complexity of this combined reduction step remains $O(m^{1.5})$.

When the graph can no longer be reduced by degree or triangle reduction, we apply color reduction, followed by clique reduction. It is known that the computation of clique reduction is relatively expensive. To achieve a better trade-off between reduction time and effectiveness, we use the bipartite lower bound before the clique reduction. Intuitively, a tighter lower bound can help reduce more vertices in the clique reduction.



$G = (V, E)$ Int k
Disjoint Lower Bound $O(n^2)$
Combined Reduction $O(m^{1.5})$
Degree Reduction $O(m)$
Triangle Reduction $O(m^{1.5})$
Color Reduction $O(nm)$
Bipartite Lower Bound $O(n^3 \log n)$
Clique Reduction $O(nT_c(n))$
Interdiction Reduction $O(n^2)$
G^*, k^*, lb Cliques

Figure 2: Flowchart of the reduction-based preprocess, time complexities are given to each step.

Algorithm 1 RECIP

Input: Graph $G = (V, E)$, an integer k
Output: $\theta(G, k)$

```
1: $G, k, lb, Cliques \leftarrow preprocess(G, k)$
2: $ans \leftarrow branc\_and\_cut(G, k, lb, Cliques)$
3: **return** $ans$
```

So far, the reductions are used in an increasing order of their complexity. Exceptions are given to the interdiction reduction. The interdiction reduction step follows the clique reduction because it asks for $\omega(G[N(u)])$ for each $u \in V$. Finally, the pre-processing outputs graph G and budget k.

4.3 Integrating the Reduction and Branch-and-cut

Our algorithm, RECIP, consists of two main components. The first component is the reduction-based preprocessing step described earlier, which aims to reduce the size of the graph. The second component is a branch-and-cut algorithm [Mitchell, 2002] to solve the ILP-CIP formulation.

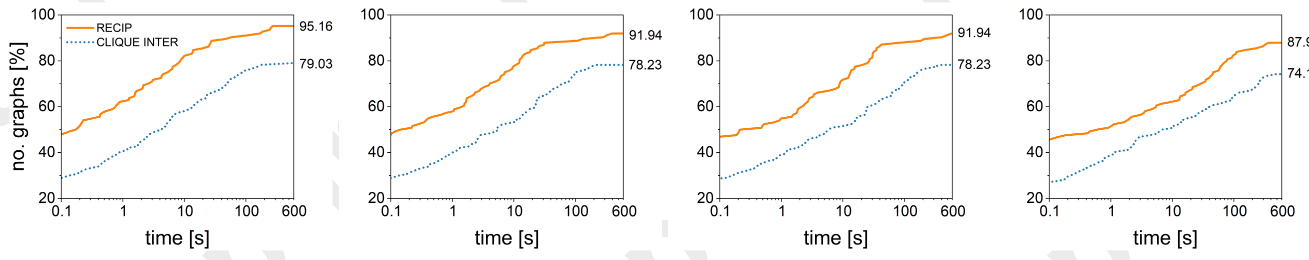
Initial Linear Program We use ILP-CIP$(G, k, \mathcal{C})$ to build the linear program, where $\mathcal{C}$ are cliques obtained by the reduction process. Additionally, based on the domination reduction rule (Lemma 9), we find all dominance relations between any u and $v \in V$. Then we add $x_u \leq x_v$ when x_u dominates x_v to tighten the LP.

Speciation Oracle When a solution (x^*, y^*) is obtained, the branch-and-cut algorithm asks a separation oracle to decide if there is a clique C such that the inequality $\sum_{u \in C} x_u^* \leq y$ is not satisfied. If so, a new inequality of the form $\sum_{u \in C} x_u \leq y$ is added to continue the search. This separation oracle is equal to the maximum clique problem in the graph $G[V^*]$ where $V^* = \{u \in V : x_u^* = 1\}$ [Furini *et al.*, 2019]. Again, we use the algorithm in [Chang, 2019] to find maximum cliques.

5 Experiments

In this section, we evaluate the performance of our algorithm. As mentioned, there are two existing methods, the general BILP solver [Becker, 2017; Tang *et al.*, 2016; Fischetti *et al.*, 2017] and CLIQUE-INTER [Furini *et al.*, 2019], for solving the CIP. According to the experiments in [Furini *et al.*, 2019], CLIQUE-INTER is faster than the BILP solver by several orders of magnitude on all tested cases. Therefore, we mainly compare our RECIP with CLIQUE-INTER in the experiments.

Experimental Setup All experiments were conducted on a machine equipped with an AMD R9-7940HX CPU and 16GB of RAM, running Ubuntu 22.04. The codes were written in C++ and compiled with GCC version 11.4.0 using the -O2 optimization flag. The IBM CPLEX solver version 12.7 was used as the underlying solver for the integer linear program. All algorithms run in single-threaded mode with the default settings of CPLEX.



(a) $k = \lceil 0.005|V| \rceil$ (b) $k = \lceil 0.01|V| \rceil$ (c) $k = \lceil 0.02|V| \rceil$ (d) $k = \lceil 0.05|V| \rceil$

Figure 3: The proportion of instances solved by both algorithms within the time range of 0.1 to 600 seconds on real-world network graphs under different k values.

Real-world Dataset Our experiments use 124 real-world networks sourced from the SNAP database and the Network Repository[Leskovec and Sosič, 2014; Rossi and Ahmed, 2015] [2]. The dataset covers a variety of categories, including social networks, technological networks, biological networks, and more. These graphs can be as large as 1.9 million vertices and 5 million edges. We leave a detailed description in the appended file.

5.1 Results on Real-world Network Graph

We compare the performance of the two algorithms under four different budget values ($k \in \{\lceil 0.005|V|\rceil, \lceil 0.01|V|\rceil, \lceil 0.02|V|\rceil, \lceil 0.05|V|\rceil\}$) with a runtime limit of 600 seconds.

Figure 3 shows the number of instances solved within different time limits for each budget value k. For every k, RECIP consistently outperforms CLIQUE-INTER across all time frames, solving over 10% more instances after 600s. Furthermore, for a fixed time frame, the number of instances solved by both algorithms decreases as k increases. This trend matches the observation that fewer vertices and edges can be reduced when k increases.

In the scatter plot in Figure 4, each point represents an instance, i.e., a pair (G, k), where the axes represent the runtime of the two algorithms. Points below the diagonal line indicate cases where RECIP is faster than CLIQUE-INTER, while points above indicate the opposite. There are 410 instances that RECIP outperforms CLIQUE-INTER, accounting for 89.7% of the total instances. We note that there are 39 instances that both algorithms cannot solve within the time limit.

Furthermore, we observed that in some relatively denser graphs, the reduction process can be highly time-consuming. For example, in graph `scc_reality`, where $|V| = 6809$ and $|E| = 4714485$, when $k = \lceil 0.05n \rceil$, the clique reduction step takes 816 seconds but only removes 5 vertices. In contrast, the subsequent branch-and-cut process completes in just 1.6 seconds.

5.2 Effectiveness of Reduction Rules

Now, we give a detailed break-up analysis of the reduction algorithms. For each instance, we record the graph size after each reduction step, i.e., degree, triangle, color, and exact clique reductions, and we also record the size of the remaining graph. Due to space limitations, we report the 10 graphs with the most vertices from those that can be solved by RECIP within 600s and where not all vertices are removed during the reduction process in Figure 5.

[2]The dataset can be downloaded from https://lcs.ios.ac.cn/~caisw/graphs.html.

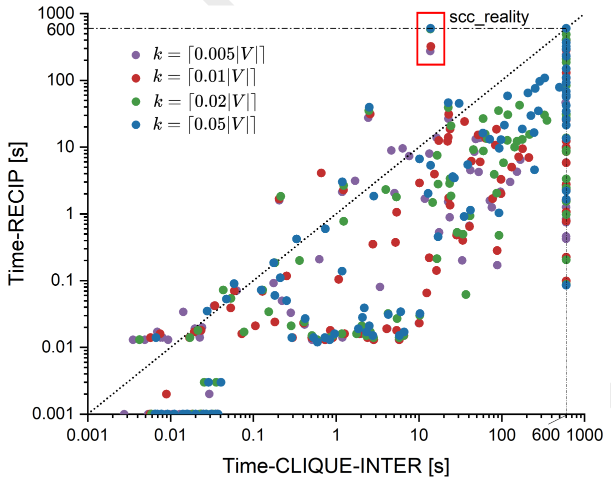


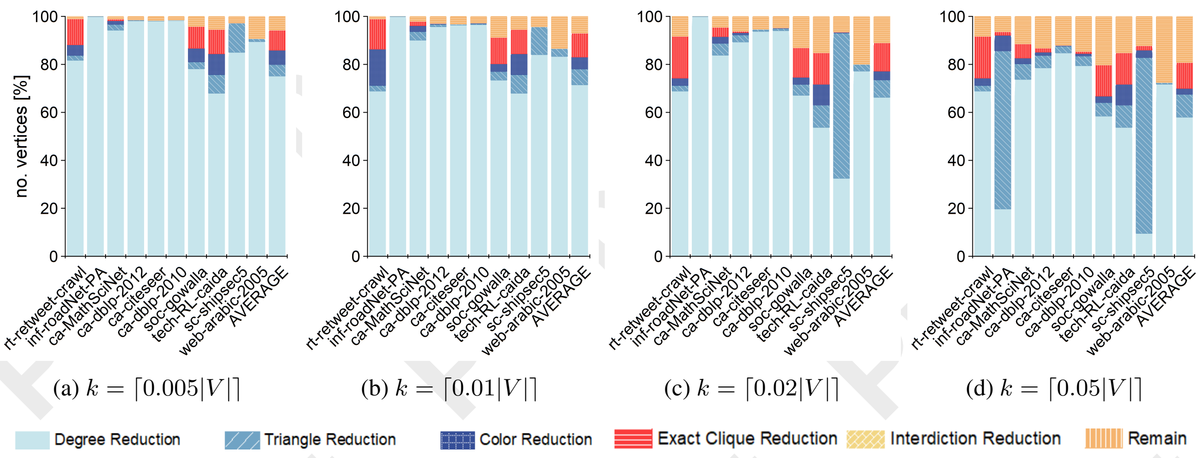
Figure 4: Runtime of both algorithms for each instance, with instances exceeding the 600-second time limit plotted at the boundary. For clarity, a dashed line representing $x = y$ is added.

Clearly, the simple degree reduction removes at least half of the vertices for the majority of graphs. This is within our expectation as these graphs are sparse. In contrast, the interdiction reduction step removes few vertices. Nevertheless, the remaining reduction steps still play an important role in reducing around 20% of the number of vertices, especially when $k = \lceil 0.05|V| \rceil$. Increasing the value of k makes the reduction process less effective, resulting in a greater number of remaining vertices.

5.3 Analysis with Lower Bounds

To demonstrate the effectiveness of our lower bound algorithm, we record the disjoint lower bound and bipartite lower bound for the 10 graphs. We present the results for $k = 0.005|V|$ in Table 1 and leave the rest in the appended files.

We observe that the gap between our lower bound algorithm and $\theta(G, k)$ is very small. The bipartite lower bound



(a) $k = \lceil 0.005|V| \rceil$ (b) $k = \lceil 0.01|V| \rceil$ (c) $k = \lceil 0.02|V| \rceil$ (d) $k = \lceil 0.05|V| \rceil$

Figure 5: Reduction proportion at each step for 10 selected graphs and the average, with different k values.

Table 1: Disjoint and bipartite lower bound for 10 selected graphs with $k = \lceil 0.005|V| \rceil$.

	disjoint		bipartite		$\theta(G, k)$
	lb	time[s]	lb	time[s]	
rt-retweet-crawl	3	0.31	3	238.10	3
inf-roadNet-PA	3	0.134	3	0.201	3
ca-MathSciNet	7	0.067	7	2.54	7
ca-dblp-2012	15	0.105	15	0.506	15
ca-citeseer	25	0.044	25	0.317	25
ca-dblp-2010	19	0.039	19	0.245	19
soc-gowalla	7	0.056	8	23.913	8
tech-RL-caida	4	0.043	4	20.212	5
sc-shipsec5	21	0.084	21	1.005	21
web-arabic-2005	49	0.051	49	4.711	49

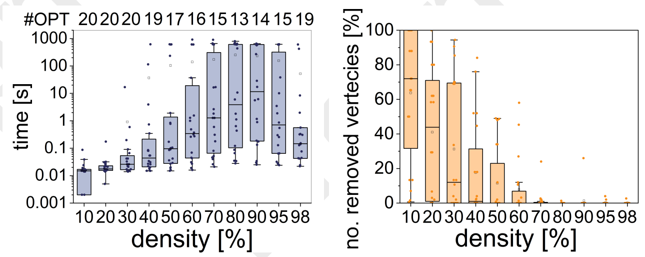
obtains a tighter bound for soc-gowalla with a sacrifice of running time. Across all the 455 instances where $\theta(G, k)$ is known, the sum of the gaps between $\theta(G, k)$ and the bipartite lower bound is only 24. This demonstrates the effectiveness of our lower-bound algorithms.

5.4 Analysis with Random Graphs

To further investigate the scalability of the algorithm, we create 5 groups of Erdős-Rényi random $G(n, p)$ graphs, each group has a vertex number $n(|V|)$ selected from $\{50, 75, 100, 125, 150\}$. In each group, we generate 11 graphs of edge densities ($p \in \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.95, 0.98\}$)[3], then we evaluate four different budget values ($k \in \{\lceil 0.05|V| \rceil, \lceil 0.1|V| \rceil, \lceil 0.2|V| \rceil, \lceil 0.4|V| \rceil\}$) for each of these graphs. The cut-off time is still 600 seconds.

In order to know how graph density affects the reduction algorithm, in Figure 6, we present the runtime and the percentage of vertices removed for different graph densities p.

[3]The edge density p is $|E|/\binom{|V|}{2}$



(a) Runtime for each instance, with the number of completed instances within the 600-second time limit indicated at the top.

(b) The percentage of vertices removed for each instance during the reduction process.

Figure 6: Results on random graphs, grouped by density.

The average runtime reaches its peak when p is in $[0.8, 0.9]$, and few vertices can be removed when $p \geq 0.7$. We guess that in these dense graphs, the reduction rules in RECIP are less effective, making it perform worse than CLIQUE-INTER on such instances. Additional evaluation on another dense benchmark from the Second DIMACS Implementation Challenge[4] confirms this trend, with RECIP and CLIQUE-INTER competing closely with each other.

6 Conclusion and Future Work

In this paper, we introduce new reduction rules and propose a novel algorithm, RECIP. Experiments demonstrate that our algorithm outperforms the current state-of-the-art method by an order of magnitude in terms of efficiency on large-scale datasets. The results highlight the effectiveness of our reduction techniques, with some limitations observed in dense graphs. Thus, it is possible to further enhance its overall performance by exploring new reduction rules for dense graphs and improving the branch-and-cut algorithm in the future.

[4]This dataset can be downloaded from https://iridia.ulb.ac.be/~fmascia/maximum_clique/

Acknowledgments

We thank the authors of CLIQUE-INTER for kindly providing the source code. This work was supported by the National Natural Science Foundation of China under grant 62372093.

References

[Abu-Khzam *et al.*, 2022] Faisal N Abu-Khzam, Sebastian Lamm, Matthias Mnich, Alexander Noe, Christian Schulz, and Darren Strash. Recent advances in practical data reduction. *Algorithms for Big Data*, pages 97–133, 2022.

[Becker, 2017] Timothy Becker. *Bilevel Clique Interdiction and Related Problems*. PhD thesis, Rice University, 2017.

[Berry *et al.*, 2004] Nina Berry, Teresa Ko, Tim Moy, Julienne Smrcka, Jessica Turnley, and Ben Wu. Emergent clique formation in terrorist recruitment. In *The AAAI-04 Workshop on Agent Organizations: Theory and Practice*, pages 1198–1208, 2004.

[Bläsius *et al.*, 2022] Thomas Bläsius, Philipp Fischbeck, Lars Gottesbüren, Michael Hamann, Tobias Heuer, Jonas Spinner, Christopher Weyand, and Marcus Wilhelm. A branch-and-bound algorithm for cluster editing. In *20th International Symposium on Experimental Algorithms (SEA 2022)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2022.

[Borgatti *et al.*, 2024] Stephen P Borgatti, Martin G Everett, Jeffrey C Johnson, and Filip Agneessens. *Analyzing Social Networks*. SAGE Publications Limited, 2024.

[Chang and Qin, 2019] Lijun Chang and Lu Qin. Cohesive subgraph computation over large sparse graphs. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 2068–2071. IEEE, 2019.

[Chang, 2019] Lijun Chang. Efficient maximum clique computation over large sparse graphs. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 529–538, 2019.

[Chen *et al.*, 2021] Xiaoyu Chen, Yi Zhou, Jin-Kao Hao, and Mingyu Xiao. Computing maximum k-defective cliques in massive graphs. *Computers & Operations Research*, 127:105131, 2021.

[Dempe, 2020] Stephan Dempe. Bilevel optimization: theory, algorithms, applications and a bibliography. *Bilevel optimization: advances and next challenges*, pages 581–672, 2020.

[Douik *et al.*, 2020] Ahmed Douik, Hayssam Dahrouj, Tareq Y Al-Naffouri, and Mohamed-Slim Alouini. A tutorial on clique problems in communications and signal processing. *Proceedings of the IEEE*, 108(4):583–608, 2020.

[Downey and Fellows, 2012] Rodney G Downey and Michael Ralph Fellows. *Parameterized complexity*. Springer Science & Business Media, 2012.

[Dunbar and Spoors, 1995] Robin IM Dunbar and Matt Spoors. Social networks, support cliques, and kinship. *Human nature*, 6:273–290, 1995.

[Fischetti *et al.*, 2017] Matteo Fischetti, Ivana Ljubić, Michele Monaci, and Markus Sinnl. A new general-purpose algorithm for mixed-integer bilevel linear programs. *Operations Research*, 65(6):1615–1637, 2017.

[Furini *et al.*, 2019] Fabio Furini, Ivana Ljubić, Sébastien Martin, and Pablo San Segundo. The maximum clique interdiction problem. *European Journal of Operational Research*, 277(1):112–127, 2019.

[Furini *et al.*, 2021] Fabio Furini, Ivana Ljubić, Pablo San Segundo, and Yanlu Zhao. A branch-and-cut algorithm for the edge interdiction clique problem. *European Journal of Operational Research*, 294(1):54–69, 2021.

[Grass and Fischer, 2016] Emilia Grass and Kathrin Fischer. Two-stage stochastic programming in disaster management: A literature survey. *Surveys in Operations Research and Management Science*, 21(2):85–100, 2016.

[Grüne and Wulf, 2024] Christoph Grüne and Lasse Wulf. Completeness in the polynomial hierarchy for many natural problems in bilevel and robust optimization. *arXiv preprint arXiv:2311.10540*, 2024.

[Hastad, 1996] Johan Hastad. Clique is hard to approximate within n/sup 1-/spl epsiv. In *Proceedings of 37th Conference on Foundations of Computer Science*, pages 627–636. IEEE, 1996.

[Hespe *et al.*, 2020] Demian Hespe, Sebastian Lamm, Christian Schulz, and Darren Strash. Wegotyoucovered: The winning solver from the pace 2019 challenge, vertex cover track. In *2020 proceedings of the SIAM workshop on combinatorial scientific computing*, pages 1–11. SIAM, 2020.

[Karp, 2010] Richard M Karp. *Reducibility among combinatorial problems*. Springer, 2010.

[Latapy, 2008] Matthieu Latapy. Main-memory triangle computations for very large (sparse (power-law)) graphs. *Theoretical computer science*, 407(1-3):458–473, 2008.

[Leskovec and Sosič, 2014] Jure Leskovec and Rok Sosič. Snap: Stanford network analysis project. https://snap.stanford.edu/, 2014. Accessed: 2025-01-12.

[Li *et al.*, 2017] Chu-Min Li, Hua Jiang, and Felip Manyà. On minimization of the number of branches in branch-and-bound algorithms for the maximum clique problem. *Computers & Operations Research*, 84:1–15, 2017.

[Luo *et al.*, 2024] Chunyu Luo, Yi Zhou, Zhengren Wang, and Mingyu Xiao. A faster branching algorithm for the maximum k-defective clique problem. In *ECAI 2024*, pages 4132–4139. IOS Press, 2024.

[Mahdavi Pajouh *et al.*, 2014] Foad Mahdavi Pajouh, Vladimir Boginski, and Eduardo L Pasiliao. Minimum vertex blocker clique problem. *Networks*, 64(1):48–64, 2014.

[Malod-Dognin *et al.*, 2010] Noël Malod-Dognin, Rumen Andonov, and Nicola Yanev. Maximum cliques in protein structure comparison. In *Experimental Algorithms: 9th International Symposium, SEA 2010, Ischia Island, Naples, Italy, May 20-22, 2010. Proceedings 9*, pages 106–117. Springer, 2010.

[Mattia, 2024] Sara Mattia. Reformulations and complexity of the clique interdiction problem by graph mapping. *Discrete Applied Mathematics*, 354:48–57, 2024.

[Mitchell, 2002] John E Mitchell. Branch-and-cut algorithms for combinatorial optimization problems. *Handbook of applied optimization*, 1(1):65–77, 2002.

[Nasirian *et al.*, 2019] Farzaneh Nasirian, Foad Mahdavi Pajouh, and Josephine Namayanja. Exact algorithms for the minimum cost vertex blocker clique problem. *Computers & Operations Research*, 103:296–309, 2019.

[Rossi and Ahmed, 2015] Ryan A. Rossi and Nesreen K. Ahmed. The network data repository with interactive graph analytics and visualization. https://networkrepository.com/, 2015. Accessed: 2025-01-12.

[Rutenburg, 1994] Vladislav Rutenburg. Propositional truth maintenance systems: Classification and complexity analysis. *Annals of Mathematics and Artificial Intelligence*, 10:207–231, 1994.

[Sageman, 2004] Marc Sageman. Understanding terror networks. *University of Pennsylvania*, 2004.

[Šikić *et al.*, 2013] Mile Šikić, Alen Lančić, Nino Antulov-Fantulin, and Hrvoje Štefančić. Epidemic centrality—is there an underestimated epidemic impact of network peripheral nodes? *The European Physical Journal B*, 86:1–13, 2013.

[Tang *et al.*, 2016] Yen Tang, Jean-Philippe P Richard, and J Cole Smith. A class of algorithms for mixed-integer bilevel min–max optimization. *Journal of Global Optimization*, 66:225–262, 2016.

[Valdez *et al.*, 2023] Lucas Daniel Valdez, Lautaro Vassallo, and Lidia Adriana Braunstein. Epidemic control in networks with cliques. *Physical Review E*, 107(5):054304, 2023.

[Van Cleemput, 2012] Katrien Van Cleemput. Friendship type, clique formation and the everyday use of communication technologies in a peer group: A social network analysis. *Information, Communication & Society*, 15(8):1258–1277, 2012.

[Wang *et al.*, 2012] Bing Wang, Lang Cao, Hideyuki Suzuki, and Kazuyuki Aihara. Impacts of clustering on interacting epidemics. *Journal of theoretical biology*, 304:121–130, 2012.

[Xiao and Nagamochi, 2017] Mingyu Xiao and Hiroshi Nagamochi. Exact algorithms for maximum independent set. *Information and Computation*, 255:126–146, 2017.

[Xiao *et al.*, 2014] Kaiming Xiao, Cheng Zhu, Weiming Zhang, Xiangyu Wei, and Songchao Hu. Stackelberg network interdiction game: nodal model and algorithm. In *The 2014 5th International Conference on Game Theory for Networks*, pages 1–5. IEEE, 2014.

[Xiao *et al.*, 2021] Mingyu Xiao, Sen Huang, Yi Zhou, and Bolin Ding. Efficient reductions and a fast algorithm of maximum weighted independent set. In *Proceedings of the Web Conference 2021*, pages 3930–3940, 2021.