KNOWLEDGE HIDING ON GRAPH DATA

A Thesis Submitted to
the Graduate School of Engineering and Sciences of
İzmir Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of

DOCTOR OF PHILOSOPHY

in Computer Engineering

by
Leyla TEKİN

December 2024
İZMİR

We approve the thesis of **Leyla TEKİN**

Examining Committee Members:

Prof. Dr. Onur DEMİRÖRS
Department of Computer Engineering, İzmir Institute of Technology

Asst. Prof. Dr. Emrah İNAN
Department of Computer Engineering, İzmir Institute of Technology

Prof. Dr. Adil ALPKOÇAK
Department of Computer Engineering, İzmir Bakırçay University

Assoc. Prof. Dr. Fatih SOYGAZİ
Department of Computer Engineering, Aydın Adnan Menderes University

Assoc. Prof. Dr. Belgin ERGENÇ BOSTANOĞLU
Department of Computer Engineering, İzmir Institute of Technology

19 December 2024

Assoc. Prof. Dr. Belgin ERGENÇ BOSTANOĞLU
Supervisor, Department of Computer Engineering
İzmir Institute of Technology

Prof. Dr. Onur DEMİRÖRS
Head of the Department of
Computer Engineering

Prof. Dr. Mehtap EANES
Dean of the Graduate School

ACKNOWLEDGMENTS

I would like to thank my thesis advisor Assoc. Prof. Dr. Belgin Ergenç Bostanoğlu, who guided me with her knowledge and experience throughout this thesis period, for her support and patience.

I would like to thank my thesis monitoring committee Prof. Dr. Adil Alpkoçak and Prof. Dr. Tolga Ayav for their valuable feedback and time.

I would also like to thank Prof. Dr. Onur Demirörs, Assoc. Prof. Dr. Fatih Soygazi and Asst. Prof. Dr. Emrah İnan for taking part in my thesis examination.

Finally, I am profoundly grateful to my precious family for the unwavering support and encouragement they have given me throughout my life.

ABSTRACT

KNOWLEDGE HIDING ON GRAPH DATA

Subgraphs and communities, which are meaningful substructures in graphs, provide important information for a deeper understanding of network structures. However, as graph analysis tools become more advanced, this introduces a new challenge, such as the risk of over-mining of knowledge from graph data. Thus, it has led to the investigation of knowledge hiding techniques. Within the scope of this dissertation, as privacy-preserving techniques for graph data, subgraph hiding and community hiding are examined. Subgraph hiding involves identifying sensitive subgraphs in a transactional graph database, and transforming the database to prevent them from being disclosed after publication of data, while preserving the original data as much as possible. Community hiding, on the other hand, involves hiding or obfuscating communities by strategically modifying the graph. With the work on community hiding that protects the privacy of users' sensitive information, such as community membership, the need for robust community detection algorithms that can effectively counter hiding becomes more evident.

In this thesis, first subgraph hiding techniques have been examined and various edge deletion-based algorithms have been proposed. Then, community hiding techniques have been investigated, and a research has been conducted to address the comprehensive overview of the techniques at three scales. Further, an algorithm has been offered for global community hiding using cliques.

ÖZET

ÇİZGE VERİSİNDE BİLGİ GİZLEME

Çizgelerde anlamlı alt yapılar olan alt çizgeler ve topluluklar, ağ yapılarının daha derinlemesine anlaşılması için önemli bilgiler sağlar. Ancak çizge analiz araçları daha gelişmiş hale geldikçe, bu durum, çizge verilerinden gereğinden fazla bilgi çıkarılması riski gibi yeni bir zorluğu da beraberinde getirir. Böylelikle, bilgi gizleme tekniklerinin araştırılmasına yol açmıştır. Bu tez kapsamında, çizge verilerinin mahremiyetini koruyan teknikler olarak alt çizge gizleme ve topluluk gizleme incelenmektedir. Alt çizge gizleme, işlemsel çizge veritabanındaki hassas alt çizgeleri tanımlamayı ve verilerin yayınlanmasından sonra onların ifşa edilmesini önlemek için veritabanını dönüştürmeyi içerirken, orijinal verileri mümkün olduğunca korur. Topluluk gizleme ise çizgeyi stratejik olarak değiştirerek toplulukların gizlenmesini veya karartılmasını içerir. Kullanıcıların topluluk üyeliği gibi hassas bilgilerinin gizliliğini koruyan topluluk gizlemeye yönelik çalışmalarla birlikte, gizlemeye etkili bir şekilde karşı koyabilecek dayanıklı topluluk tespit algoritmalarına olan ihtiyaç daha da belirgin hale gelir.

Bu tezde öncelikle alt çizge gizleme teknikleri incelenmiş ve çeşitli kenar silme tabanlı algoritmalar önerilmiştir. Daha sonra, topluluk gizleme teknikleri araştırılmış ve tekniklerin üç ölçekte kapsamlı bir şekilde ele alınması için bir araştırma yapılmıştır. Ayrıca, klikler kullanılarak küresel topluluk gizleme için bir algoritma önerilmiştir.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

CS . . . . . Community Structure
EDH . . . . . Edge Deletion-Based Heuristic
4clqCH . . . . . 4-clique Community Hiding
FSM . . . . . Frequent Subgraph Mining
CAM . . . . . Canonical Adjacency Matrix
DFS . . . . . Depth-First Search
SBM . . . . . Stochastic Block Model
CPM . . . . . Clique Percolation Method
FSH . . . . . Frequent Subgraph Hiding
HF . . . . . Hiding Failure
AP . . . . . Artifact Patterns
IL . . . . . Information Loss
Dist . . . . . Distance
NMI . . . . . Normalized Mutual Information
ARI . . . . . Adjusted Rand Index
ONMI . . . . . Overlapping Normalized Mutual Information
CommS . . . . . Community Splits
CommU . . . . . Community Uniformity
MNMI . . . . . Modified Normalized Mutual Information
VI . . . . . Variation of Information
SJD . . . . . Split-Join Distance
CC . . . . . Constant Community
AE . . . . . Attack Efficiency
FGA . . . . . Fast Gradient Attack
GCN . . . . . Graph Convolutional Network
SNMF . . . . . Symmetric Non-negative Matrix Factorization
DICE . . . . . Disconnect Internally, Connect Externally
CDA . . . . . Community Detection Attack
DBA . . . . . Degree-Based Attack
MPL . . . . . Maximization of Persistence Loss

GNN . . . . . Graph Neural Network

SAEP . . . . . Self-Adaptive Evolutionary Deception

DFP . . . . . Degree First Deception

GAE . . . . . Graph Auto-Encoder

ST . . . . . Sanitization Table

DAG . . . . . Directed Acyclic Graph

UMST . . . . . Union of Maximum Spanning Trees

CHAPTER 1

INTRODUCTION

Complex networks are inside nearly every domain of daily life, including social networks to analyze human relationships, such as friendships, professional ties, and shared interests, transportation networks of urban and national systems to enhance service delivery, population networks to monitor and control disease transmission, and biological networks that reveal gene and protein interactions to drive advancements in genomics. These networks are represented using graph structures composed of nodes and edges. In these models, nodes symbolize entities within the network, and edges signify the connections or interactions between them.

Analyzing the substructures of a graph, such as its subgraphs, is an effective method for understanding the graph because subgraphs are the fundamental components of the graph. The following examples of their widespread usages in different applications can be given: For anomaly detection, subgraphs play a crucial role in identifying patterns that are indicative of anomalies, like fraudulent activities (*1*). They can be utilized to identify patterns that define functional connections in protein-protein interaction networks or to discover common features in molecular compounds for drug development (*2*). They can enhance classification of structural brain graphs (*3*). Further, they are applied in graph clustering (*4–6*). In summary, subgraphs demonstrate local structural characteristics and are valuable in many graph mining tasks.

Communities, on the other hand, are other substructures that appear in a network graph. A community is a group of nodes that have dense internal connections but relatively sparse connections to the rest of the network. Community detection is the process of identifying the community structure (CS), which in turn provides valuable insights into the underlying dynamics and organizational patterns within the network. Numerous algorithms have been developed across various disciplines for the community detection problem. As discussed in the study (*7*), the applications of community detection are widespread: it enables identifying social media users with similar views or behaviors, finding terrorist groups by analyzing the extended social connections, optimizing retail strategies,

segmenting customers for tailored recommendations, revealing hidden relationships between research publications, and predicting financial trends in stock market data.

As graph analysis tools have been improved, a new challenge has emerged, namely over-mining of knowledge on graph data (*8*). That is why, the issue of preserving privacy of sensitive knowledge has become increasingly important. In this thesis, two problems of knowledge hiding on graph data (**Subgraph Hiding** and **Community Hiding**) are examined.

Publishing a graph database involves privacy risks since some knowledge in the database, which may be sensitive for the database owner, may be learned upon publication. **Subgraph Hiding**, as the first studied problem of this thesis, is applied to facilitate publishing while preserving the disclosure of sensitive subgraphs. For this, first subgraph mining techniques are applied to the database. Sensitive subgraphs are identified among the knowledge obtained as a result of subgraph mining. In order to prevent the sensitive subgraphs from being obtained through similar subgraph mining techniques in the database to be published, an appropriate transformation (i.e., sanitization) process is defined. There are two parameters that are taken into consideration for the transformation process: First, the transformation process must ensure that it eliminates sensitive subgraphs, and secondly, it must preserve the originality of the database as much as possible. The former one shows whether the transformation process is correct or not, and the later one shows the quality of the transformation process. The transformation process can be done through various operations. Most often, this involves modifying or deleting some data in the database so that it does not contain knowledge designated as sensitive. The modification operation may result in the creation of some information in the database that did not exist in its original state and therefore is not real. Unlike modification, deletion is generally a preferred method because it does not produce unreal information (*9*, *10*).

Community Hiding, as the second problem of this thesis, is a privacy-preserving task that involves concealing or obfuscating communities by imperceptibly altering connections. For example, if some members in a community work for the same organization, it is likely that other community members are also affiliated with this organization. Such a data breach could cause consequences such as targeted advertising. Therefore, to preserve user privacy from malicious community detection, community hiding is studied (*11*). Another example is that some groups, such as activists and police, collaborate on social

networks like Twitter or Facebook but do not want to be detected by detection tools, so they strategically manage their connections to avoid detection. Community hiding techniques can be used maliciously, such as by terrorists seeking to communicate covertly. It highlights the need for new community detection algorithms that can effectively counter (i.e., that are robust to) deception (*12*, *13*).

Community hiding means launching attacks against community detection algorithms to degrade their performance and reduce the accuracy of their results. Hence, it is also known as 'community detection attack'. This problem is defined in three distinct scales in the study (*14*) as (i) global attack (global community hiding), (ii) target community attack (target community hiding) and (iii) target node attack (target node hiding). That is, they aim to maximize the change in community structure, hide a target community, or hide target node(s), respectively, by modifying the minimum number of edges in the graph.

1.1. Contribution of the Thesis

This thesis aims to give a contribution to the field of knowledge hiding on graph data, specifically "Subgraph Hiding" and "Community Hiding". The contributions can be explained as follows:

For **Subgraph Hiding**, five different edge deletion-based algorithms are proposed, which are Edge Deletion-based Heuristic (EDH) algorithm, EdgeDegree algorithm, Matchings & EdgeDegree algorithm, EdgeEntropy algorithm and Matchings & EdgeEntropy algorithm. The EDH algorithm developed as part of this thesis was published (*15*). The idea of the EDH algorithm is to select the most frequent edge in the matches of a sensitive subgraph pattern inside a graph as the victim edge to be deleted. Removing the most frequent edge from the graph eliminates most of the matches of the sensitive subgraph. Then, the EdgeDegree algorithm aims to improve the EDH. It chooses the most frequent edge with 'minimum sum of node degrees' in the matches. Further, the Matchings & EdgeDegree algorithm chooses the victim edge by considering the multiple patterns' matches. If sensitive patterns share a common edge, removing this edge deletes the matches of multiple patterns in one step. Lastly, the concept of graph entropy is applied for subgraph hiding problem and the edge entropy is offered in the process of choosing the victim edge. By leveraging the edge entropy, the EdgeEntropy and Matchings & EdgeEntropy

algorithms are proposed.

For **Community Hiding**, first existing community hiding algorithms are investigated. In the literature, there is no survey that includes all of the hiding (i.e., attack) scales. Hence, a review is written to cover all of them. For each hiding scale, strategies are examined and a summary table is formed to provide the key characteristics associated with each strategy. After completing the exhaustive review on community hiding (community detection attacks), an article (*16*) has been prepared and published. Furthermore, for global community hiding, an algorithm, namely 4-clique Community Hiding (4clqCH), is offered. The 4clqCH algorithm removes the most common edges in the four-cliques of the graph in order to disrupt the cliques, thereby changing the communities extracted from them. Although there are community hiding algorithms (*17*, *18*) to attack against overlapping community detection algorithms, they hide target nodes in the overlapping areas. There is no study that examines global community hiding for overlapping community detection algorithms that allow nodes to be in more than one community as in the real-world. The effectiveness of the 4clqCH algorithm against such algorithms is analyzed.

1.2. Organization of the Thesis

Chapter 2 first provides the basic graph concepts to understand the research presented in this thesis. The chapter then delves into two critical tasks in graph mining: Subgraph Mining and Community Detection. Following this, the concept of graph hiding is explored, positioned as the complementary counterpart to graph mining. The chapter examines two core tasks within graph hiding, Subgraph Hiding and Community Hiding. Subsequently, the chapter presents the evaluation measures used to assess the effectiveness of graph hiding techniques. Lastly, the chapter concludes with an exploration of subgraph entropy, a key approach for privacy preservation in graph-based data.

Chapter 3 presents a comprehensive review of the related work on Subgraph Hiding and Community Hiding. The problem of Community Hiding has been studied in the literature at three distinct scales of attack. These include the target node attack, which aims to hide the community of a specific node within a graph; the target community attack, which seeks to conceal a particular community; and the global attack, which is designed to disrupt the overall community structure of the network.

Chapter 4 introduces the algorithms developed and proposed in this thesis for addressing the problem of graph hiding. The chapter begins with the algorithms designed for subgraph hiding problem, which include the Edge Deletion-based Heuristic (EDH) algorithm (*15*), EdgeDegree algorithm, Matchings & EdgeDegree algorithm, EdgeEntropy algorithm, and Matchings & EdgeEntropy algorithm. Following this, the chapter shifts its focus to the proposed global community hiding algorithm, 4-clique Community Hiding (4clqCH), which is designed as the global attack to destroy the overall community structure within a graph.

Chapter 5 focuses on the performance evaluation of the proposed methods for graph hiding, offering an analysis of their effectiveness. For the subgraph hiding section, the evaluation begins by benchmarking the EDH algorithm against three existing vertex masking algorithms to establish its relative performance. Subsequently, a comparative analysis is conducted among the edge deletion-based subgraph hiding algorithms introduced in this thesis, including EDH, EdgeDegree, Matchings & EdgeDegree, EdgeEntropy and Matchings & EdgeEntropy. In the community hiding section, the evaluation centers on the proposed 4-clique Community Hiding (4clqCH) algorithm. Its performance is compared with that of the existing community hiding algorithms, particularly in terms of their ability to deceive community detection algorithms and effectively obscure community structures.

Chapter 6 concludes the thesis by summarizing the key findings from the proposed algorithms and their performance evaluations for the fields of subgraph hiding and community hiding. Furthermore, the chapter gives potential avenues for future research in these fields.

CHAPTER 2

BACKGROUND

This chapter introduces the foundational concepts necessary to understand the research covered in this thesis. It begins with the basic graph theory terminology. It then proceeds with two key tasks within graph mining, Subgraph Mining and Community Detection. The chapter further explores graph hiding, which is considered as the other side of the same coin. Two fundamental tasks of graph hiding, Subgraph Hiding and Community Hiding, are examined. Next, the measures employed to evaluate the effectiveness of graph hiding methods are presented, offering insight into how their performance is assessed. Finally, the last section of this chapter presents the subgraph entropy for privacy preservation on graphs.

2.1. Basic Graph Terminology

A **graph** consists of a set of n vertices (or nodes), denoted as V, and a set of m edges, denoted as E, which represent the relationships or connections between the vertices. If the edges of a graph do not have a specific direction, it is an **undirected** graph. Otherwise, it is a **directed** graph. For instance, social networks like Facebook, where connections are mutual and symmetric (i.e., both users agree to be friends), can be represented as undirected graphs. In contrast, social networks like Twitter can be modeled as directed graphs, where relationships are not necessarily mutual. One user follows another, but the reverse may not be true.

If the vertices and/or edges of a graph are assigned labels from a symbol set, then such a graph is called a **labeled** graph. An **unlabeled** graph does not have these assignments. Figure 2.1a and 2.1b illustrate graphs whose vertices and edges are labeled. In a **weighted** graph, the edges are associated with the weights (i.e., numerical values) that typically reflect certain properties of the connections, such as cost, capacity, or distance. If no weights are assigned to the edges, or all edges carry the same weight, the graph is considered **unweighted**. Whether a graph is weighted or unweighted depends on the

specific characteristics of the problem it represents.

A **connected** graph is one in which a path exists between every pair of vertices, allowing traversal from one vertex to another via a sequence of edges. In undirected graphs, this implies that all vertices belong to a single connected component. In the case of directed graphs, the graph is termed **strongly connected** if a directed path links any vertex to every other vertex, and **weakly connected** if it would be connected by ignoring the direction of the edges.

A graph dataset may take the form of either a **single, large graph** integrating all vertices and edges within one structure, or it may consist of multiple smaller graphs. In the latter case, where the dataset is composed of several graph transactions, it is referred to as a **transactional graph database**. This distinction is critical for various graph mining applications, where the focus may shift between examining a single graph and exploring a collection of smaller, separate graphs.

Given graphs H and G, **subgraph isomorphism** can be explained as follows; These graphs are isomorphic if there exists a bijection f between their vertex sets (specified as f: $V(H) \rightarrow V(G)$). That is, in graph H, any two vertices, u and v, are adjacent if and only if their corresponding vertices $f(u)$ and $f(v)$ are adjacent in graph G. Additionally, when the graphs have vertex and/or edge labels, the labels of vertices u and $f(u)$ and/or the labels of edges (u,v) and $(f(u),f(v))$ must be equal. The problem of subgraph isomorphism checks whether a graph contains a subgraph that is isomorphic to another graph. Figure 2.1 depicts an example for subgraph isomorphism.

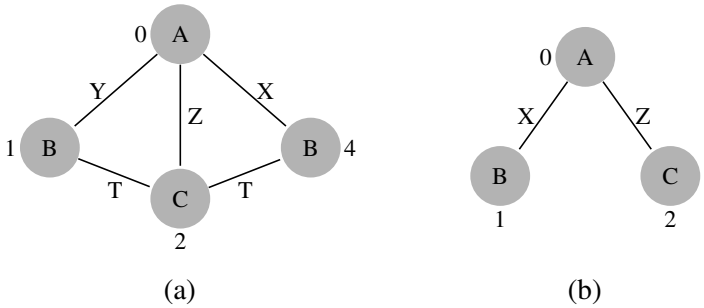


Figure 2.1. An example of subgraph isomorphism.

An **induced** subgraph means that it includes all the edges linking the corresponding vertices of the graph. In Figure 2.1, the subgraph (2.1b) is not an induced subgraph of the graph (2.1a) because there is no edge between nodes 1 and 2 in the subgraph as the correspondence of the edge $(4, 2)$ of the graph.

2.2. Graph Mining

Data mining refers to the uncovering meaningful and valuable knowledge from data through the use of specific algorithms and tools (*19*). As the volume of data grows exponentially in the digital age, there is an increasing need to extract precise and relevant information from this vast amount of data, making data mining a vital area of research.

The data involved in data mining can come from a variety of sources and exist in different forms, such as text, audio, graphs, and images. Graphs, in particular, are effective at modeling complex structures and their relationships, making them widely applicable across many fields. In social networks (*20*), for example, graphs are used to depict users as nodes and their relationships as edges. Similarly, in chemical structures (*21*), atoms and their bonds are represented as graph nodes and edges, respectively. Biological networks (*2*, *22*) leverage graphs to map proteins and their interactions, while in wireless sensor networks (*23*), graphs model the communication between sensor motes.

Graph mining has grown into a pivotal and thriving domain within data mining, offering profound insights into graph-based data. It allows for the discovery of hidden patterns, detection of anomalies, and identification of key individuals or communities within complex networks. For instance, recommendation systems can leverage graph mining techniques to suggest new movies to users by identifying preferences shared by similar users, thereby enhancing user experience and personalization. Graph mining encompasses various tasks, such as discovering frequent subgraph patterns, node/graph classification, clustering, community detection, link prediction, and other forms of analysis. Within the scope of this thesis, two graph mining tasks are investigated: Subgraph Mining and Community Detection. They are the tasks for the other side of the picture, where the idea is to hide the information that can be discovered by them.

2.2.1. Subgraph Mining

One key task in graph mining is frequent subgraph mining (FSM), which is concerned with discovering subgraphs that occur frequently within a given graph dataset. Identifying recurring structures within data is essential since these patterns can reveal alternative perspectives on the underlying data. The extracted frequent patterns can further

be used in subsequent tasks like clustering, classification, constructing graph indices, and enabling similarity searches within graph databases (*24*).

Definition 1 (Frequent Subgraph Mining) *Let D be a graph dataset (a single large graph or a transactional graph database with multiple graphs) and let σ be a predefined minimum support threshold. Frequent subgraph mining finds all frequent subgraphs in the dataset D such that the support of each of such subgraphs is greater than or equal to the support threshold σ.*

The method for calculating the support σ of a subgraph ($0 < \sigma < 1$) in the dataset varies by the dataset setting. In the *multiple-graph setting*, the support of the subgraph refers to the ratio of the number of graphs in which it appears to the number of graphs in the dataset, that is, it ignores how frequently the subgraph occurs in each individual graph. Conversely, in the *single graph setting*, the support of the subgraph means the number of occurrences of the subgraph within that graph.

Before mining the frequent subgraphs, the input graph data is transformed into a properly encoded form. Adjacency matrix and adjacency list are among the most widely adopted methods for graph representation. Since multiple adjacency matrices (or adjacency lists) may represent the same graph, ensuring a unique representation of graphs is essential for efficient subgraph isomorphism testing. So, canonical labeling methods like canonical adjacency matrix (CAM) and minimum DFS (Depth-First Search) code have been proposed.

The FSM process is composed of two primary phases. The initial phase of FSM begins with the generation of candidate subgraphs—potential frequent subgraphs—that will then be checked for frequency evaluation. Different strategies can be used for candidate generation, such as level-wise join, extension, etc. In the level-wise join strategy, two subgraphs of size k are combined to generate a new subgraph of size $(k + 1)$. For this join operation to be feasible, both k-size subgraphs must share a common subgraph of size $(k - 1)$. The main drawbacks of this strategy are that multiple candidates can be produced by one join operation and duplicate candidates can be produced due to distinct join operations. Alternatively, the extension strategy involves creating a new connected subgraph of size $(k + 1)$ by adding an additional edge to a k-size subgraph. The issue with this strategy is that subgraph extension can occur at multiple nodes, which may result in the creation of unnecessary candidates. A specialized form of this strategy, the right-most

path extension strategy, restricts the addition of new edges specifically to nodes on the right-most path.

Following the candidate generation, the next phase in the FSM process is to count occurrences of each candidate subgraph within the graph dataset. Frequency counting often represents the most computationally demanding part in subgraph mining; hence, search space pruning and redundancy elimination are typically employed to filter out non-promising candidates prior to counting. The frequency counting stage yields the frequency of each candidate subgraph generated, which is then evaluated whether it is frequent or not. However, as mentioned above, frequency counting of a subgraph varies based on the type of the graph dataset.

Frequent subgraph mining algorithms are primarily divided into two categories according to their candidate generation methods: the Apriori approach and the pattern-growth approach. The Apriori approach employs the level-wise candidate generation strategy, requiring all frequent subgraphs of size $(k-1)$ to be known to generate the candidate subgraphs at the next iteration. The pattern-growth approach expands a frequent subgraph by adding an edge, bypassing the need to generate and test all possible candidates. AGM (*25*), FSG (*26*), and HSIGRAM (*27*) algorithms follow the Apriori approach, and FFSM (*28*), gSpan (*29*), MoFa (*30*), CloseGraph (*31*) and VSIGRAM (*27*) algorithms are examples for pattern-growth apprach.

Graphs employed in the field of FSM may be directed or undirected, and they may also be labeled or unlabeled. The objective of a subgraph mining algorithm can vary, which may be extracting all frequent subgraphs in a graph dataset or finding a specific subset of the frequent subgraphs, such as closed or maximal subgraphs. Furthermore, different FSM algorithms can be designed according to the dynamicity of the graph structure (i.e., static or dynamic graph data). This diversity offered for various characteristics allows the development of tailored FSM algorithms depending on the specific needs and constraints of the underlying data and objectives.

2.2.2. Community Detection

Complex networks often have an inherent community structure, being considered one of their characteristics. A community (i.e., cluster or group) within a network is

expressed as a subset of nodes that are densely interconnected with one another, while they have relatively sparse links to nodes outside of this subset. Community detection refers to the task of revealing and identifying these cohesive substructures within the network, hence serving as a foundational step in understanding the organization and patterns of connectivity and obtaining insights into the functional interactions that arise from these groupings.

Definition 2 (Community Detection) *Consider a graph $G = (V, E)$, where V is the set of nodes (vertices) and E is the set of edges. The community detection problem determines the community structure indicated by $\overline{C}$, that is, it partitions the graph nodes V into K distinct communities, such that $\overline{C} = \{C_1, \ldots, C_K\}$, with each subset $C_i \subseteq V$ corresponding to a specific community.*

Networks are categorized as either static or dynamic, based on whether their structure remains fixed or changes over time. For static networks, community detection aims to reveal the underlying community structure, while for dynamic networks it requires approaches that can account for the evolving nature of communities, addressing temporal changes to accurately capture their changing structures. Since community hiding algorithms have not yet been developed in dynamic networks, this study focuses on methods developed for static networks. Within the realm of static network analysis, community detection methods are typically classified into (i) traditional methods, (ii) spectral methods, (iii) optimization methods, (iv) statistical methods, and (v) dynamics-driven methods (*32*, *33*).

Traditional methods include 'partitional clustering, graph partitioning, hierarchical clustering', which are the basic approaches for community detection, and their general characteristics can be given as follows. Partitional clustering splits the graph vertices into a predefined number of clusters (i.e., K) through optimizing a given cost function with respect to the distances, with the widely known k-means algorithm and its many variations exemplifying this approach (*34*, *35*). Graph partitioning, on the other hand, segments the vertices into K groups of a certain size by minimizing inter-group edges, as demonstrated by the Kernighan-Lin algorithm (*36*). In networks with hierarchical structures, multiple levels of clusters can exist. Hierarchical clustering techniques can either be agglomerative or divisive. Notably, Girvan and Newman's divisive algorithm (*37*) iteratively removes edges with the highest betweenness (shortest path) to uncover community structure. Spectral methods, meanwhile, utilize the spectral properties of graph-associated matrices, such

as the adjacency or Laplacian matrices, to reveal community structure, as illustrated in references (*38–41*).

Optimization-based methods in community detection aim to maximize or minimize an objective function that evaluates the partitioning quality. Modularity is the most prevalent measure used for this purpose; however, maximizing modularity is NP-hard (*42*), which has spurred the creation of various approximation algorithms, including greedy approaches, genetic algorithms, and simulated annealing techniques (*43*). Newman (*44*) introduces a hierarchical agglomerative algorithm that optimizes modularity using a greedy strategy, and Clauset et al. (*45*) further improve its computational efficiency through strategic enhancements and sophisticated data structures. To refine modularity optimization within spectral methods, Newman (*46*) proposes a series of refinement procedures. Additionally, integer programming has been explored to address challenges in community detection (*42*). Among the notable modularity-based algorithms, the Louvain method proposed by Blondel et al. (*47*) applies a two-phase iterative process to maximize modularity. Other prominent algorithms within this category include the Leiden (*48*) and Combo (*49*) algorithms.

Statistical approaches to community detection are generally based on probabilistic models that interpret network data through a generative framework. The stochastic block model (SBM) is the most widely employed generative model for community-structured graphs. Dynamics-driven methods leverage dynamic processes occurring within the network—such as random walks, spin dynamics, and synchronization—to identify communities. Methods based on random walks, for example, assume that nodes with similar properties are more likely to cluster within the same community. In this context, Pons and Latapy's Walktrap method (*50*) employs a distance matrix generated from random walks to group nodes. Rosvall and Bergstrom's Infomap algorithm (*51*) minimizes the description length of a random walk. Some techniques also draw on models from statistical mechanics, such as the spin glass model (*52*), where a Hamiltonian function is optimized. Additionally, Raghavan et al. (*53*) propose the label propagation algorithm, an efficient and straightforward method in which each node initially holds a unique label that is iteratively updated based on the majority label of neighboring nodes.

Communities can be categorized into two distinct types: overlapping communities and disjoint (non-overlapping) communities. Overlapping communities enable nodes to

belong to more than one groups simultaneously, reflecting the real-world scenarios where individuals often join several groups at the same time. But, non-overlapping communities are strictly partitioned, with each node allocated to a single community, thus creating distinct and mutually exclusive groups. A well-known method for detecting overlapping communities is the Clique Percolation Method (CPM) (*54*), operating on the principle of overlapping cliques, where a clique is defined as a complete subgraph. The methods (*55–58*) are other examples for overlapping community detection.

So far, the attention has primarily been directed towards examining conventional community detection techniques since community hiding algorithms try to fool them. Nevertheless, the field has advanced notably, particularly through the parallelization of such conventional techniques to enhance their effectiveness and practical usability (*59*, *60*). Furthermore, deep learning has also shown impressive results in the field of community detection (*61*).

2.3. Graph Hiding

Graph mining extracts meaningful and valuable knowledge from graph data. In the domain of graph mining, a range of critical tasks is studied to extract knowledge, including community detection, frequent subgraph mining, link prediction, node classification, anomaly detection. The knowledge extracted from a graph mining task may represent hidden patterns, relationships, or structures within the network that reveal insights.

While graph mining is used to extract the knowledge, graph hiding tries to degrade the performance of graph mining algorithms. This may arise from the need for privacy protection (*62*). For instance, social media users who prefer to keep certain undisclosed connections private may seek heuristic strategies to confound link prediction algorithms, preventing the revelation of sensitive relationships (*63*). Similarly, individuals may wish to obscure their profiles from being linked through node similarity measures, thereby evading analyses that rely on such measures (*64*). Additionally, certain groups may attempt to avoid identification as tightly connected communities to conceal group affiliation and shared interests (*12*, *13*). In another case, techniques are applied to obscure the origin of network diffusion against source detection algorithms (*65*). Moreover, some methods aim to protect specific sensitive subgraphs within graph data, ensuring that critical structures remain

hidden from mining processes (*9*, *15*). Through these techniques, graph hiding seeks to counter knowledge extraction efforts, promoting the privacy and security of individuals and organizations within complex networks.

Within the scope of this thesis, subgraph hiding and community hiding, which are two subdivisions of graph hiding, are investigated.

2.3.1. Subgraph Hiding

Subgraph hiding enables graph data to be shared with external parties or used in collaborative environments without compromising the privacy or confidentiality of subgraph structures that may contain sensitive information. Hence, it strikes a balance between data utility and privacy preservation, thereby addressing the growing demand for secure data sharing in graph-based applications. This problem has been studied for transactional graph databases. Let us start by introducing the terminology to explain subgraph hiding:

Definition 3 (Sensitive pattern) *Given a graph database D, a set of sensitive patterns, denoted by SP, is determined by the owner of the graph data. In the work* (*9*)*, SP is chosen from the frequent subgraph patterns FP, that is SP⊂FP, where FP is extracted by mining on the database D based on the support threshold* σ.

Definition 4 (Disclosure threshold) *It is a hiding threshold indicated by* ψ. *All the sensitive patterns in the set SP are hidden according to the defined disclosure threshold. The frequency of each sensitive pattern P is decreased below it, so freq(P)* $< \psi$.

Definition 5 (Sensitive graph) *A graph is a sensitive graph if it contains any sensitive pattern from SP. Formally, let G be a graph in a database D, and SP be the set of sensitive patterns. For any* $P \in SP$, *if P is a subgraph of G, then G is called a sensitive graph.*

The problem of frequent subgraph hiding (FSH) converts a graph database D into a sanitized graph database D' by satisfying the following constraints: (i) hiding all the sensitive patterns, that is, the frequency of each sensitive pattern is less than the disclosure threshold in the sanitized database D' and (ii) preserving the non-sensitive frequent patterns, that is, each non-sensitive pattern that was frequent in D should be frequent in D' as much

as possible. Thus, the objective of FSH is to modify the given database to get the sanitized one such that when a frequent subgraph mining tool is executed on D', it does not reveal any sensitive patterns, while minimizing its effect on the non-sensitive patterns and original data. Figure 2.2 depicts the process of frequent subgraph hiding. Its inputs are a graph database D and a set of sensitive patterns SP along with a disclosure threshold ψ. It outputs the sanitized database D'. Now, D' can be shared since if a subgraph mining algorithm is used on D', only non-sensitive patterns can be discovered.

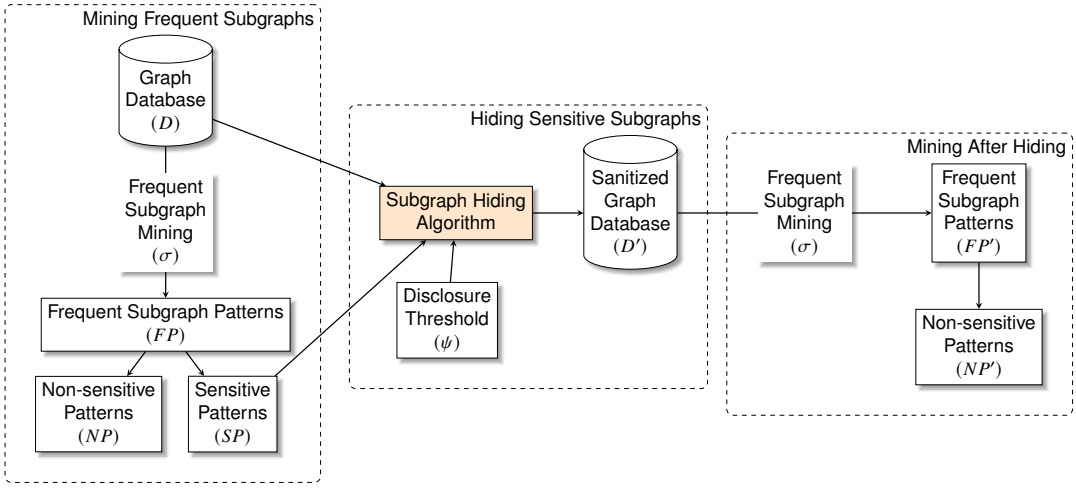


Figure 2.2. The process of frequent subgraph hiding.

To solve the FSH problem, two subproblems need to be addressed. The first subproblem is identifying sensitive graphs to be sanitized in the database for every sensitive pattern. The second one is to sanitize a determined sensitive graph so that it does not support the sensitive pattern anymore.

2.3.2. Community Hiding

Community detection algorithms can be significantly perturbed when only a small fraction of network links is allowed to be modified, as each link plays a distinct role in preserving the community structure. A community hiding algorithm attempts to deceive community detection algorithms and can be seen as a symmetrical process on the other side of the wall. This process involves strategic adjustments to the network structure, such as edge and/or node update operations, with the objective of fooling the detection algorithms in the identification of the community structure, so that obscuring the correct community affiliations and interactions in the network. Consequently, the hiding algorithms

aim to achieve a targeted level of obfuscation by making changes to the network structure according to the number of allowed connections.

Definition 6 (Budget) *It represents the upper limit on the number of edge modifications that a community hiding algorithm is allowed to execute within the network. This restriction constrains the extent of adversarial changes applied to the graph structure to ensure that the hiding is undetectable. Budget is symbolized as β.*

Definition 7 (Rewiring) *It refers to a specific edge operation in which an edge linked to a node is deleted, followed by the addition of another edge to the same node, leaving the total degree of the node unchanged. This technique subtly changes the connections of the node to other nodes while maintaining its overall connectivity, hence reducing the effect of hiding in the network structure.*

Figure 2.3 presents an overview of the community hiding process. Initially, the community structure is detected from the original graph through a particular community detection algorithm. Following this, a community hiding algorithm is applied to the graph that executes update operations (i.e., edge operations, such as addition, deletion, and rewiring, and/or node operations, such as addition, deletion, and moving) constrained by a predefined budget. The modified graph is then re-evaluated using the community detection algorithm, which may reveal notable alterations in the resulting community structure. For example, nodes that were a part of a single community may become dispersed across different communities, or the number of communities may change substantially. It should be noted that this figure serves as a conceptual overview; specific community hiding algorithms may leverage varying types of knowledge or require additional inputs.

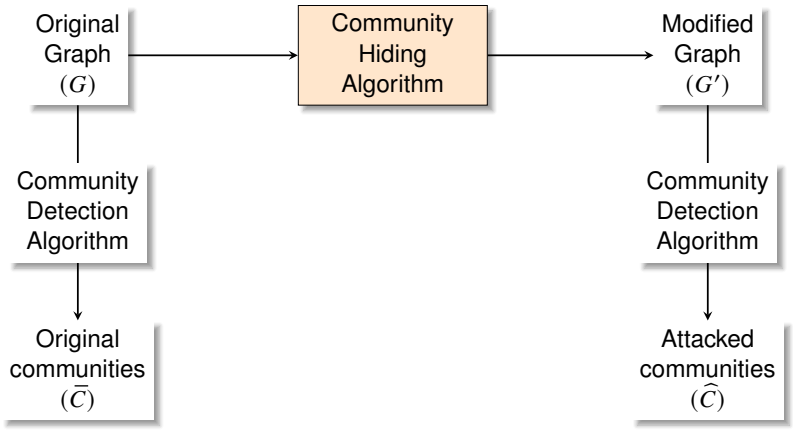


Figure 2.3. The process of community hiding.

Section 3.2 details the specific knowledge utilized by each algorithm. Throughout this thesis, 'community hiding' and 'community detection attack' can be used interchangeably.

2.4. Measures used in Graph Hiding

This section summarizes the measures for evaluating the performance of graph hiding methods. The first subsection gives the measures for subgraph hiding, and the second subsection presents the measures for community hiding.

2.4.1. Measures for Subgraph Hiding

The Frequent Subgraph Hiding (FSH) problem has some side-effects (*66*), which are outlined below. To assess the efficiency of an FSH algorithm, these side-effects can be measured.

- **Hiding Failure (*HF*).** The amount of sensitive patterns that are mined after sanitization. It is computed as;

$$HF = |SP'| \big/ |SP|, \tag{2.1}$$

  where $|SP'|$ and $|SP|$ represent the number of sensitive patterns that are found on the output (sanitized) database D' and the original database D, respectively.

- **Artifact Patterns (*AP*).** The number of the identified patterns that are artifacts. For some vertices, they include the masking symbol as the vertex label. It is calculated as;

$$AP = |FP'| - |FP \cap FP'|, \tag{2.2}$$

  where $|FP'|$ and $|FP|$ are the number of frequent patterns that are found on the output database D' and the original database D, respectively.

- **Information Loss (*IL*).** The ratio of non-sensitive patterns that are hidden unintentionally during the hiding process. It is computed as;

$$IL = \big((|FP| - |SP|) - (|FP \cap FP'| - |SP'|)\big) \big/ (|FP| - |SP|), \tag{2.3}$$

  where it filters out the frequent patterns that include the masking symbol by the

$|FP \cap FP'|$. This formula is derived from calculation of the side-effects in the reference (*67*). The loss of a frequent pattern containing a sensitive pattern is not taken into account as it will be hidden once the sensitive one is hidden (*67*).

- **Distance (*Dist*).** The total number of vertices and edges that are altered (masked/deleted) due to the sanitization, and calculated as;

$$Dist = |V| - |V'_U| + |E| - |E'_U|, \tag{2.4}$$

where V'_U is the set of the vertices unchanged in D', and E'_U is the set of the edges unchanged in D'. It is formalized to assess the distance of subgraph hiding algorithms following different approaches by utilizing graph concepts.

2.4.2. Measures for Community Hiding

This section outlines the key performance evaluation measures commonly employed to assess the effectiveness of community hiding algorithms. The measures detailed below capture various aspects of the hiding algorithms, including partition quality, similarity between two partitions, and hiding success.

- **Modularity:** To evaluate the quality of a network division, the modularity—originally introduced by the study (*68*)—is defined as $Q = \sum_i (e_{ii} - a_i^2)$. Here, e_{ii} represents the proportion of edges within community C_i that connect nodes within the same community, while $a_i = \sum_j e_{ij}$ denotes the fraction of edges connected to nodes in C_i. Essentially, modularity assesses the difference between the observed count of intra-community edges and the expected count if edges were randomly assigned. Higher modularity values indicate a better community structure. This measure is particularly useful for networks with unknown community structure. For weighted networks, the modularity formulation is extended (*69*).

- **Normalized Mutual Information (NMI):** This measure quantifies the similarity between two community partitions (X and Y) through an information-theoretic lens. Here, $H(X)$ represents the entropy associated with partition X, while $I(X, Y)$ denotes the mutual information between the partitions, capturing the extent to which one

partition informs us about the other. It is calculated as $NMI = \frac{2I(X,Y)}{H(X)+H(Y)}$. This expression can be applied to the community structures ($\overline{C}$ and $\widehat{C}$) detected before and after a hiding process, respectively, as follows:

$$NMI = \frac{-2\sum_{i=1}^{|\overline{C}|}\sum_{j=1}^{|\widehat{C}|} m_{ij}\log\left(\frac{m_{ij}n}{M_i M_j}\right)}{\sum_{i=1}^{|\overline{C}|} M_i \log\left(\frac{M_i}{n}\right) + \sum_{j=1}^{|\widehat{C}|} M_j \log\left(\frac{M_j}{n}\right)}, \tag{2.5}$$

where m represents the confusion matrix with rows indicating the actual communities and columns indicating the found communities, m_{ij} is the number of nodes shared between the actual community i and found community j, M_i is the sum of elements in row i, M_j is the sum of elements in column j, and n is the total number of nodes in the network (i.e., the sum of all matrix elements). A higher NMI score reflects a greater degree of similarity between the two community partitions (*70*).

- **Adjusted Rand Index (ARI):** It quantifies the similarity between two partitions by using a pair-counting approach. It is formally defined as follows (*71*):

$$ARI = \frac{\sum_{ij}\binom{m_{ij}}{2} - \left[\sum_i \binom{M_i}{2}\sum_j \binom{M_j}{2}\right] / \binom{n}{2}}{\frac{1}{2}\left[\sum_i \binom{M_i}{2} + \sum_j \binom{M_j}{2}\right] - \left[\sum_i \binom{M_i}{2}\sum_j \binom{M_j}{2}\right] / \binom{n}{2}}. \tag{2.6}$$

  The ARI has a value range $[-1, 1]$. Higher ARI values indicate greater similarity between the two partitions being compared.

- **Overlapping NMI (ONMI):** It is a measure used to compare overlapping communities (*72*), which is defined as $ONMI = \frac{I(X,Y)}{max(H(X),H(Y))}$.

- **Success rate:** It measures the proportion of target nodes that are incorrectly clustered, relative to the total number of target nodes (*73*), or the percentage with which the target node is assigned to a community other than its original one when the process is conducted repeatedly (*74*).

- **AML:** The average number of link modifications required to successfully hide a target node (*73*).

- **Percentage degree increase:** It refers to the percentage of the target node's degree increase as a result of the hiding (*14*).

- **Community retention probability:** It represents the likelihood that the target nodes remain within their initial communities following the execution of the hiding (*75*).

- **Miss ratio:** It is expressed as $FN/|V - C|$, where C denotes the target community and FN represents the proportion of nodes of the target community that are assigned to a different community. This measure indicates the proportion of the target community nodes within the network segment where they try to hide (*76*).

- **Concealment measure (M):** It evaluates the effectiveness of hiding a target community C inside a community structure $\overline{C}$. Two measures, $M^{'}$ and $M^{''}$, are proposed to capture different aspects of this concealment process. $M^{'}$ evaluates how well the members of C are dispersed across the communities in $\overline{C}$, while $M^{''}$ quantifies the extent to which C is hidden within the crowd (*12*).

$$M = \alpha M^{'} + (1 - \alpha) M^{''} \tag{2.7}$$

$$M^{'} = \frac{|\{C_i \in \overline{C} : C_i \cap C \neq \emptyset\}| - 1}{\max(|\overline{C}| - 1, 1) \max\limits_{C_i \in \overline{C}}(|C_i \cap C|)} \tag{2.8}$$

$$M^{''} = \sum_{C_i \in \overline{C}} \frac{|C_i \setminus C|}{\max(n - |C|, 1)} \tag{2.9}$$

  where α is a parameter that is in the interval $[0, 1]$ and n is the number of nodes in the graph.

- **Community deception score (H):** Fionda and Pirro (*13*) propose three key indicators of good hiding of a target community C: (i) *reachability preservation*, the members of C should reach one another, meaning that modifications should not disrupt the connectivity; (ii) *community spread*, the members of C should distribute across as many communities within the network as possible; and (iii) *community hiding*, the members of C should be positioned within the largest possible communities. The deception score H is introduced to capture these three indicators. Let C be a target community and $\overline{C} = \{C_1, C_2, \ldots, C_k\}$ be a community structure, the score H is defined as:

$$H(C,\overline{C}) = \left(1 - \frac{|S(C)| - 1}{|C| - 1}\right) \times \left(\frac{1}{2}\left(1 - \max_{C_i \in \overline{C}}\{R(C_i, C)\}\right) + \frac{1}{2}\left(1 - \frac{\sum_{C_i \cap C \neq \emptyset} P(C_i, C)}{|C_i \cap C \neq \emptyset|}\right)\right), \tag{2.10}$$

where $|S(C)|$ refers to the count of connected components in the subgraph generated by the members of community C, and recall (R) and precision (P) of a community detection algorithm A_D in relation to C are formally defined as follows:

$$R(C_i, C) = \frac{\#\, C \text{ members in } C_i}{|C|}\ \forall C_i \in \overline{C}. \tag{2.11}$$

$$P(C_i, C) = \frac{\#\, C \text{ members in } C_i}{|C_i|}\ \forall C_i \cap C \neq \emptyset. \tag{2.12}$$

The objectives (i), (ii), and (iii) outlined above are achieved by the left factor, the first term of the right factor, and the second term of the right factor, respectively. Now, let us consider whether the deception score H can be directly applied (via maximizing it) to address the target community hiding problem. The score incorporates the knowledge of the community structure $\overline{C}$, and therefore, it would require the knowledge of the community detection algorithm A_D that generated $\overline{C}$. It means that the deception would be dependent on A_D.

- **Community splits (CommS):** It quantifies the number of communities within the modified network that include the members of the target community (*77*).

- **Community uniformity (CommU):** It quantifies how the target community members are distributed across the communities in the modified network by utilizing the entropy concept (*77*).

- **Modified NMI (MNMI):** In large networks, the concealment of a target community may not substantially affect those communities that do not share direct connections with it. Therefore, this measure assesses the NMI between the community memberships of nodes in the target community and their directly linked neighbors before and after the hiding process. Its range is consistent with that of NMI (*77*).

- **Fitness:** The proposed fitness function can be utilized to evaluate the hiding effect (*14*).

- **Variation of information (VI):** It is a measure used to compare two partitions, X and Y, based on the principles of information theory. It is defined as (*78*):

$$VI = (H(X) - I(X,Y)) + (H(Y) - I(X,Y)). \tag{2.13}$$

  A lower VI value indicates a higher similarity between the two partitions.

- **Split-join distance (SJD):** It quantifies the distance between two community partitions, denoted as X and Y (*79*). It is defined by $SJD(X,Y) = PD_Y(X) + PD_X(Y)$, where the $PD_Y(X)$ represents the projection distance of partition Y from partition X and computed as follows: For each community in Y, identify the community in X with which it shares the maximum overlap. The summation of these maximal overlap sizes is then subtracted from the number of elements in Y. A smaller SJD value indicates a higher similarity between the partitions.

- **Jaccard index:** It is used for comparing two community partitions (X and Y) (*80*). It is defined as $J(X,Y) = |X \cap Y|/|X \cup Y|$.

- **Recall:** It is used for comparing two community partitions (X and Y) (*80*). It is calculated as $R(X,Y) = |X \cap Y|/|X|$.

- **Precision:** It is used for comparing two community partitions (X and Y) (*81*). It is computed as $P(X,Y) = |X \cap Y|/|Y|$.

- **Node-centric measures:** It involves the increase in the graph safeness and the reduction in the graph persistence (*82*). The equations for node safeness and persistence (which is quite similar to the permanence) are presented in Section 3.2.2, enabling the computation across the graph by summing the node-based values.

- **Constant community (CC) measures:** Constant communities represent groups of nodes appearing consistently within the same community for different community detection algorithms. The measures found from CCs include the number of nodes that form CCs, the average density of these communities, and the average hub dominance (*82*).

- **Attack efficiency (AE):** It quantifies the effectiveness of a hiding. It is the number of incorrectly grouped nodes with a limited number of edge alterations (β). It is

defined as (*83*):

$$AE = \frac{\text{Number of altered nodes}}{\beta}. \tag{2.14}$$

- **BN (β * NMI):** It refers to the hiding cost. It quantifies the number of edge alterations necessary to reduce the NMI value. Its lower value indicates more effective hiding (*84*).

2.5. Subgraph Entropy for Privacy Preservation on Graphs

Let us begin with describing the Shannon's entropy (*85*). The entropy of a set of probabilities $(p_1, p_2, ..., p_n)$ is defined as follows:

$$H = -\sum_{i=1}^{n} p_i \log p_i. \tag{2.15}$$

To introduce information-theoretical graph measures, a tuple $(\lambda_1, \lambda_2, ..., \lambda_n)$ of non-negative integers is offered to form a probability distribution (*86*)

$$p_i = \frac{\lambda_i}{\sum_{j=1}^{n} \lambda_j} \quad i = 1, 2, ..., n. \tag{2.16}$$

Hence, the entropy can be indicated as

$$H = -\sum_{i=1}^{n} p_i \log p_i = \log\left(\sum_{i=1}^{n} \lambda_i\right) - \sum_{i=1}^{n} \frac{\lambda_i}{\sum_{j=1}^{n} \lambda_j} \log \lambda_i. \tag{2.17}$$

Qiao et al. (*87*) propose a centrality measure to quantify the influence of nodes in networks. This measure uses an entropy centrality model and can be utilized to detect the vital nodes. It computes the influence of a node by calculating the local influence of it on one-hop neighbors and the indirect influence of it on two-hop neighbors, and summing up their weighted values.

To find the local influence of a node v_i, first a subgraph consisting of the node v_i and its one-hop neighbors is considered. For each node in the subgraph, the SDC_i is calculated as

$$SDC_i = \sum_{j=1}^{M} b_{ij}, \tag{2.18}$$

where M indicates the number of one-hop neighbors of node i, and b_{ij} represents the existence of the edge between node i and node j. The value of b_{ij} is 0 or 1. The tuple is defined as $(\lambda_1, \lambda_2, ..., \lambda_{M+1})$ and $\lambda_i = SDC_i$. Thus, the local influence of node i (LI_i) is

$$LI_i = \log\left(\sum_{i=1}^{M+1} SDC_i\right) - \sum_{i=1}^{M+1} \frac{SDC_i}{\sum_{j=1}^{M+1} SDC_j} \log SDC_i. \tag{2.19}$$

If the node i has a two-hop neighbor node k, N_{ik} is the number of common neighbors of these nodes. Assume that node j is one common neighbor of them. The LI_i and LI_j have already been calculated. If $N_{ik} = 1$, the indirect influence of node i on k (II_{ik}) is

$$II_{ik} = I_{ij} \times I_{jk} = LI_i \times LI_j. \tag{2.20}$$

With the assumption that each path from node i to node k has the equal weight, it will be written as

$$II_{ik} = \sum_{k=1}^{N_{ik}} \frac{LI_i \times LI_j}{N_{ik}}. \tag{2.21}$$

The indirect influence of node i (II_i) is written by considering all of its two-hop neighbors as follows:

$$II_i = \frac{\sum_{k=1}^{M_i} II_{ik}}{M_i}, \tag{2.22}$$

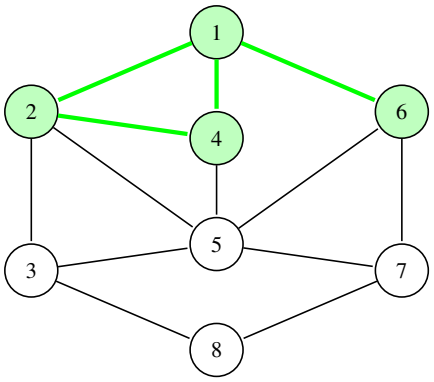
where M_i is the number of two-hop neighbor nodes of the node i. After calculating the local influence and indirect influence, the total influence of node i is computed as

$$I_i = w_1 LI_i + w_2 II_i, \tag{2.23}$$

where w_1 and w_2 are the weights and $w_1 + w_2 = 1$.

An example graph is demonstrated in Figure 2.4 to point out the overall influence of node 1. The subgraph containing node 1, its one-hop neighbors, and the edges between any two of them are shown with green color. Additionally, it shows the SDC_i values of the nodes in the subgraph.

By setting the logarithm base to 10, the local influence is found using the Equation 2.19 as follows:



Node	SDC_i
1	3
2	2
4	2
6	1

Figure 2.4. An example graph to show the influence of node 1. The SDC_i values of the nodes in the subgraph.

$$LI_1 = \log_{10}\left(\sum_{i=1}^{4} SDC_i\right) - \sum_{i=1}^{4} \frac{SDC_i}{\sum_{j=1}^{4} SDC_j} \log_{10} SDC_i = 0.5737.$$

The indirect influence of node 1 on its two-hop neighbors (i.e., nodes 3, 5, and 7) is found by the Equation 2.22 as

$$II_1 = \big(LI_1 \times LI_2 + (LI_1 \times LI_2 + LI_1 \times LI_4 + LI_1 \times LI_6)/3 + LI_1 \times LI_6\big)/3 = 0.3584.$$

Sen et al. (*88*) introduce subgraph entropy as an information-theoretic metric for human brain graphs. Additionally, node and edge entropies that are the types of subgraph entropies are used to rank the regions (nodes) and edges in a brain graph. It is stated that the important nodes (or edges) are the ones with the maximum entropy in subgraphs. A node (a brain region) in a functional brain graph is associated with one mean time-series, and an edge weight represents the absolute value of correlation coefficient between the mean time-series of the vertices.

Node entropy of a node v_i is calculated from the subgraph consisting of the node v_i and its one-hop neighbors. Let V_{v_i} be the set of nodes in the subgraph and E_{v_i} be the one-hop edges from v_i. Let

$$q'_{km} = \begin{cases} \dfrac{e'_{km}}{\sum(e'_{km} \in E_{v_i})} & \text{if } k \neq m, \\ 0 & \text{otherwise,} \end{cases} \tag{2.24}$$

where e'_{km}'s are the normalized edge values and q'_{km}'s are the normalized values in the

adjacency matrix. The node entropy is defined as

$$H = -\sum_{\substack{k,m \in V_{v_i} \\ q'_{km} \neq 0}} q'_{km} \log_2 q'_{km}. \tag{2.25}$$

Edge entropy of an edge e_{ij} is found from the subgraph which combines the one-hop subgraphs of the nodes v_i and v_j. Its vertex set is $V_{e_{ij}} = (V_{v_i} \cup V_{v_j})$ and edge set is $E_{e_{ij}} = (E_{v_i} \cup E_{v_j})$. Then,

$$q'_{km} = \begin{cases} \dfrac{e'_{km}}{\sum(e'_{km} \in E_{e_{ij}})} & \text{if } k \neq m, \\ 0 & \text{otherwise,} \end{cases} \tag{2.26}$$

where q'_{km}'s are the normalized values in the adjacency matrix. The edge entropy is defined as

$$H = -\sum_{\substack{k,m \in V_{e_{ij}} \\ q'_{km} \neq 0}} q'_{km} \log_2 q'_{km}. \tag{2.27}$$

An example graph of calculating the edge entropy is illustrated in Figure 2.5a. The subgraph containing the edge (1,2) is given in Figure 2.5b. The edge values are normalized so that their sum becomes 1. Its edge entropy is calculated as $-[0.6 \times \log_2 0.6 + 0.2 \times \log_2 0.2 + 0.1 \times \log_2 0.1 + 0.1 \times \log_2 0.1] = 1.5710$.

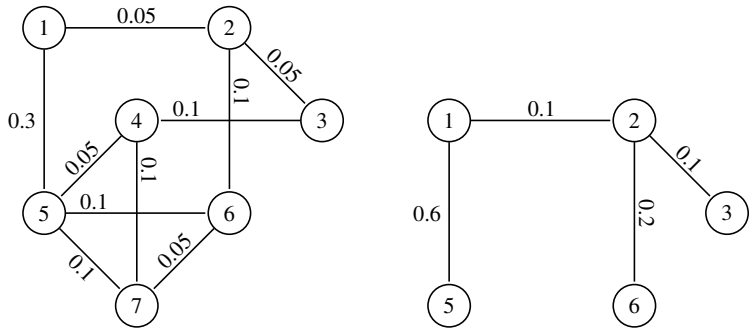


Figure 2.5. An example graph of calculating the edge entropy.

CHAPTER 3

RELATED WORK

This chapter presents a detailed review of the existing literature in the fields of Subgraph Hiding and Community Hiding to offer a thorough understanding of prior work. In the domain of Community Hiding, studies have primarily focused on three distinct scales of attack, each targeting a specific aspect of community detection (*14*, *16*). The first is the target node attack, which aims to obscure the community membership of a specific node. The second is the target community attack, which seeks to conceal a specific community. The third scale is the global attack, which adopts a broader approach, aiming to disrupt the overall community structure of the network.

3.1. Subgraph Hiding

Before delving into subgraph hiding, first research on association rule hiding and frequent itemset hiding is explored to gain a better understanding of solutions for tabular data. Their approaches can be categorized into five different classes: 1) Border Based Approaches (*89–92*) concentrate on adjusting the boundaries within the lattice of frequent and non-frequent itemsets in the dataset. 2) Exact Approaches (*93–95*) formulate the problem as a constraint satisfaction problem and solve it using integer programming. While this approach guarantees optimal solutions, it requires significant computational resources. 3) Reconstruction Based Approaches (*96*, *97*) rebuild the sanitized database using the non-sensitive frequent itemsets. 4) Evolutionary Approaches (*98–100*) are suggested as solutions for the problem of sensitive pattern hiding using genetic algorithms, ant colony system algorithms, etc. 5) Heuristic Approaches use fast and efficient algorithms with two main subgoals in the hiding process: concealing as many sensitive patterns as possible and minimizing side effects. However, they do not guarantee the optimality. There are two heuristic approaches, which are distortion-based (*66*, *101–105*) and blocking-based (*106*, *107*) approaches. Blocking approaches change the real values in the selected transactions to unknowns (like question marks) without adding false information to the database, while

distortion approaches change 0s to 1s or vice versa in the selected transactions.

Abul and Gökçe (*9*) introduce the concept of subgraph hiding. To hide sensitive subgraphs, they propose three blocking-based algorithms that mask the labels of some vertices. While these algorithms can effectively hide sensitive subgraphs, they generate fake subgraphs, affect all edges of the masked vertices, and may potentially compromise privacy. Their approach chooses sensitive graphs with fewer matches for sanitization and gives three heuristics for the local sanitization of a sensitive graph to remove the matches of a sensitive pattern: Heuristic1, Heuristic2, and Heuristic3.

- *Heuristic1*: It identifies unique vertex labels in the sensitive pattern and then for each label determines the number of vertices in the graph with that label. It then masks the vertices having the label that appears the least in the graph.

- *Heuristic2*: It detects the most frequent vertex in all matches of the sensitive pattern. The label of this vertex is then masked in the graph, and the matches that include this vertex are removed from the match set. It is repeated until no matches remain.

- *Heuristic3*: The frequency of each unique graph vertex in all matches of the sensitive pattern is computed. The frequencies found are sorted in decreasing order. The first i vertices are selected for masking, ensuring that the sum of their frequencies is equal to or greater than the match set size. However, the chosen vertices might be in the same matches, and the process from the beginning is repeated until no matches are left. It may need to execute a subgraph matching algorithm multiple times.

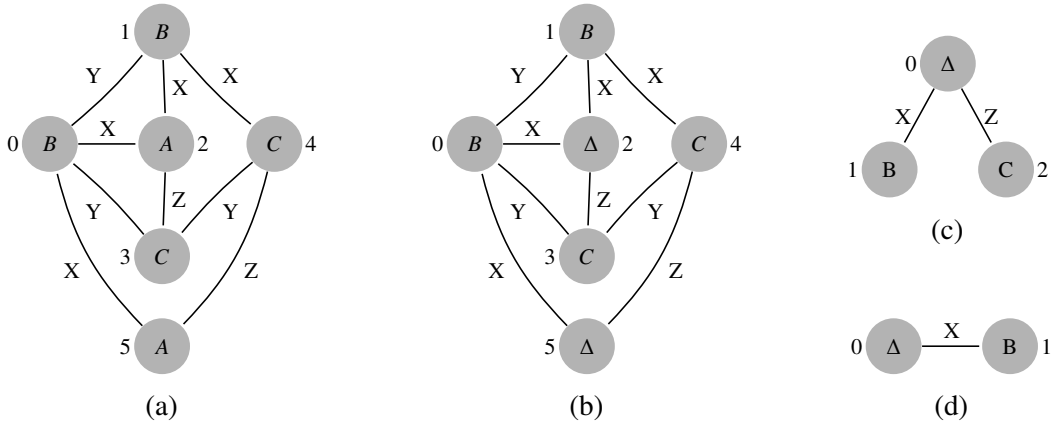


Figure 3.1. A local sanitization example with Heuristic2; (a) a sensitive graph containing three matches of the sensitive pattern given in Figure 2.1b, (b) the sanitized graph, and (c)-(d) potential artifact patterns.

Figure 3.1 shows an example of sanitizing a sensitive graph with Heuristic2. Suppose that the subgraph in Figure 2.1b is considered as the sensitive pattern and the sensitive graph in Figure 3.1a is provided. There are three matches of the pattern in this graph: $\{2{:}0, 0{:}1, 3{:}2\}$, $\{2{:}0, 1{:}1, 3{:}2\}$, and $\{5{:}0, 0{:}1, 4{:}2\}$. Heuristic2 selects the most frequent vertex in the matches. If there is more than one such vertex, it chooses randomly. The vertex 2 is one of the most frequent ones in the matches, appearing in the two matches. So, Heuristic2 masks it in the graph. Then, there is one match left after removing its matches. Heuristic2 chooses the vertex 5 and masks it. Figure 3.1b displays the resulting sanitized graph. This masking technique may create artifact patterns with missing vertex labels, as depicted in Figure 3.1c and Figure 3.1d.

3.2. Community Hiding

Community detection attacks try to deteriorate the performance of community detection algorithms. In this section, the problem of community detection attacks is given in three scales; target node, target community, and global attacks (*16*). The available attack algorithms in the literature for each scale are explained. Some algorithms are designed for multi-scale problems like EPA (*14*) or CH-SNMF (*75*); the algorithm explanation is given in the subsection where it first appears in these cases.

3.2.1. Target Node Attack

In certain cases, a specific node within a network may prefer to avoid detection as part of any particular community. This node may have affiliation with sensitive or private groups, such as political or religious groups, or may not want its group to be known due to personal preferences. The main goal of a target node attack is to obscure the community membership of a target node within a network, ensuring that the community it belongs to cannot easily be detected. This can be executed by altering the connections around the target node, making it harder to accurately place the node within its original community.

Definition 8 (Target Node Attack) *Given a target node u, suppose it is a member of community C_i in the original graph $G = (V, E)$. The target node attack problem is to avoid the target node u from being identified as part of its initial community C_i by altering the*

connections around it. After the attack, suppose that it is included in C_j in the adversarial graph G', which is as different as possible from C_i. That means the target node is clustered into the wrong community.

Figure 3.2 illustrates an example of a target node attack on Zachary's Karate network (*108*). The community structure, comprising four distinct communities, is found using the Louvain algorithm (*47*). Node 19 is chosen as the target node, with its border highlighted in red. Initially, it belongs to the blue community indicated by the circle. Given an attack budget of 2, the target node removes its connection with node 1, which is in the same community, and establishes a new connection with node 29 from a different community. That means edge $(19, 1)$ is deleted within the community of the target node and edge $(19, 29)$ is added between communities. After the attack, when the Louvain is re-applied to the modified network, as shown on the right, the target node is assigned to a different community, marked with plum color and hexagon shape. The goal here is to determine the optimal structural modification to the neighborhood of the target node. This modification should ensure that the target node remains hidden. It is considered successful when the original community and the new community involving the target node are different.

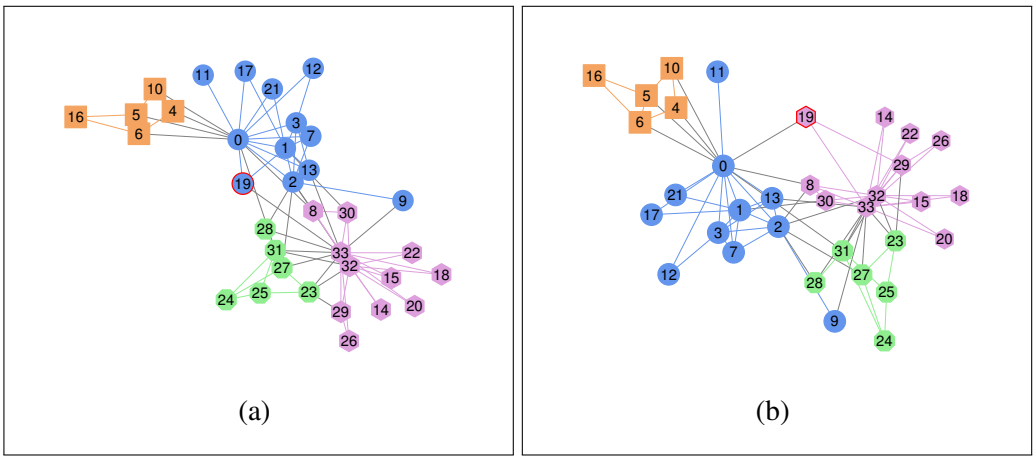


Figure 3.2. Illustration of a target node attack. (a) Community structure of the original graph with the target node border highlighted in red. (b) Community structure identified after applying a target node attack.

Chen et al. (*73*) put forward fast gradient attack (FGA) on network embedding to avoid target nodes from being identified (i.e., wrongly clustered in community detection). FGA comprises two parts: (i) generation of the adversarial network based on the graph

convolutional network (GCN) gradient and (ii) using the adversarial network to attack the GCN or other network embedding methods. In the first stage, for a target node, a target loss function is computed. The partial derivative of it concerning each element of the adjacency matrix is calculated. Then, the link with the maximum absolute gradient is chosen to add/delete since it affects the result of the target node more. In the experiments, the K-means algorithm is used to obtain the community detection results from the embedding vectors.

The problem of community detection attacks is formalized in three scales, global, target community, and target node attacks, by the study (*14*). Each attack problem is modeled as an optimization algorithm, and a genetic algorithm-based EPA method is proposed. The index (ID) of a link (added or deleted link) is used as a gene. Each chromosome represents an attack. For a target node attack, the fitness function is based on only the degree change $f = \Psi(d')$, where $\Psi(d') = exp(c \times d')$, c is a constant to control decay speed, $d' = d/|E|$, $d = \frac{1}{4}\sum_{i=1}^{n} |d_i - \bar{d_i}|$, n is the number of nodes in the graph, and d_i and $\bar{d_i}$ are the degrees of node i before and after the attack. Unequal crossover is adopted so that the chromosome length can be changeable. In the mutation process, network structural properties (i.e., between pairs of two nodes, betweenness for deletion, and the shortest path lengths for addition) are used.

Bernini et al. (*74*) address the problem of community membership hiding. The goal of the node deception task is to disassociate a target node u from its original community C_i. The objective is defined by setting a threshold for the similarity between the target node's original community C_i and new community C_i' by excluding it, that is, $sim(C_i - \{u\}, C_i' - \{u\}) \leq \tau$, where $\tau \in [0, 1]$. The loss function is formulated as $loss = l_{decept} + \lambda l_{dist}$. The first part evaluates to 1 if node u remains in the community C_i or 0 otherwise (the goal is achieved). The second part assesses the distance between two graphs (G and G') and their corresponding community structures ($\overline{C}$ and $\widehat{C}$). The target node can delete edges within its community and add new edges to nodes outside its community. The hiding problem is solved using deep reinforcement learning, and the advantage actor-critic framework is used (*109*).

A unified framework (CH-SNMF) is introduced for the three scales of community hiding (*75*). It utilizes the clustering properties of symmetric non-negative matrix factorization (SNMF). Rather than constructing the feature matrix, the adjacency matrix A is used. Matrix U = U_{ij} (community indicator) specifies the probability of a node v_i being

part of community C_j. $min||\mathrm{A} - \mathrm{UU}^T||_F^2$ is obtained such that $\mathrm{U} \geq 0$. At each iteration, nodes having a high membership are chosen when updating the matrix U. For rewiring, it removes the edges between nodes with high membership in a community and creates edges to nodes with high membership that belong to other communities.

Table 3.1 summarizes and compares target node attack algorithms. For each study, reference of the study, target node attack algorithm name(s), update operation, whether intra-/inter-commmunity edges are used or not, knowledge needed (if any prior information required), and measures used to compare the attack algorithms with the algorithms in the comparison column are specified in the table. When the table is examined, it can be seen that the EPA algorithm employs only edge addition, while other methods incorporate both edge addition and removal to modify the network's structure. Rewiring is only applied in CH-SNMF. Considering the community of the target node and the external communities, the algorithms that delete the intra-community edges and add inter-community edges are DRL-Agent and CH-SNMF. The aim is concealing a specific node in target node attacks, but all attack algorithms utilize the entire network. Some algorithms, such as DRL-Agent and CH-SNMF, even rely on prior knowledge of communities. Although success rate is used as a performance measure in two studies, different measures are also preferred. Comparisons of the algorithms are typically made against random baselines.

Table 3.1. Summary of target node attacks.

Ref.	**Community Hiding**	**Update**	**Intra/ Inter**	**Knowledge**	**Measure**	**Comparison**
(*73*)	FGA	EDel, EAdd	✗	G	Success rate, AML	Rand, Nettack, DICE
(*14*)	EPA	EAdd	✗	G	Percent. degree increase	Rand
(*74*)	DRL-Agent	EDel, EAdd	✓	G, CS	Success rate, NMI	Rand, Degree, ROAM
(*75*)	CH-SNMF	EDel, EAdd (Rewire)	✓	G, $\|CS\|$	Comm. retention prob.	ROAM

3.2.2. Target Community Attack

There may be scenarios where a group, such as police enforcement or activists, needs to interact and cooperate in a network without disclosing their community membership. A target community attack, also known as community deception, involves disguising a specific community that may be private or sensitive. This attack aims to conceal the existence of a target community within the network. For this, the connections of members of the target community are altered, which complicates the detection. Changing the connections of the target community may spread its members to other communities, causing the target community to become obscure.

Definition 9 (Target Community Attack) *Given a target community $C \subseteq \overline{C}$, identified by a community detection algorithm f on the original graph $G = (V, E)$, the target community attack problem is to hide the target community C by altering the connections of its members within a specified budget β. After the attack, C is no longer detectable by the community detection algorithm f. That is, in the adversarial graph G', the nodes in C are distributed among a set of new communities, denoted by $\widehat{C} = \{C'_1, C'_2, \ldots, C'_L\}$.*

An illustrative example for a target community attack on the Karate network (*108*) is depicted in Figure 3.3. The community structure identified with the Louvain algorithm on the original network consists of four communities. Assume that the community whose members are shown as octagons and bordered in red is targeted, that is, target community members are $\{23, 24, 25, 27, 28, 31\}$. Considering that the update budget is 4 and the following updates are applied to the graph (deleting edges $(23, 27)$ and $(24, 31)$, and adding edges $(24, 4)$ and $(25, 4)$), the community detection result on the updated network is obtained, as shown on the right, pointing out (*13*) (i) its members are evenly distributed in two communities, (ii) its members are better 'hidden' within the larger communities, and (iii) all its members are reachable from one another.

Nagaraja (*76*) first introduced the concept of hiding a community to counter community detection algorithms by allowing the community to alter its structure. The study considers only two communities where the nodes of the hidden community try to enter the main network (i.e., the other community). Their strategies are only based on link additions to the nodes with high centrality values, chosen from each of the two communities.

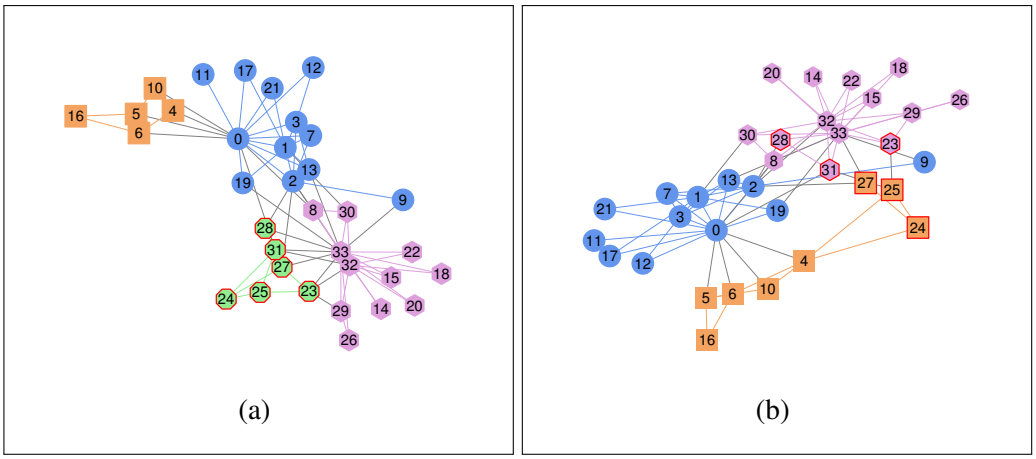


Figure 3.3. Illustration of target community attack. (a) Community structure of the original graph with the borders of target community nodes highlighted in red. (b) Community structure after applying a target community attack.

Waniek et al. (*12*) also focus on how a community can disguise itself to reduce the likelihood of being discovered through community detection. A heuristic algorithm named "disconnect internally, connect externally" (DICE) is proposed, which works by randomly deleting the links between members of the target community (intra-community edges) and adding the links between members and non-members of the target community (inter-community edges). The DICE algorithm is inspired by modularity. It does not require knowing the entire network topology and can be applied by any group of people.

Fionda and Pirro (*13*) model community deception as an optimization problem and propose "community safeness". Before giving it, firstly node safeness $\sigma(\mu, C)$ of node μ in G is defined to measure hiding of it in the community C as follows:

$$\sigma(\mu, C) = \frac{1}{2}\frac{|V_C^u| - |E(u, C)|}{|C| - 1} + \frac{1}{2}\frac{|\tilde{E}(u, C)|}{deg(u)}, \tag{3.1}$$

where $V_C^u \subseteq C$ is the set of nodes of C reachable from u by passing only via nodes in C, $E(u, C)$ is the set of internal edges of node $u \in C$, $\tilde{E}(u, C)$ is the set of external edges of node $u \in C$, and $deg(u)$ is the degree of node u. Node safeness of u considers two factors related to the indirect connections of the node u with the nodes in the community C and the external connections of the node u. That is, the first factor explains how well u transmits information in C. The second factor explains how u is concealed in the network with respect to its degree. The second one is used to confuse detection because it leads to a better community deception if u is diverse in terms of its edges. Moreover,

the community safeness $\sigma(C)$ is defined as $\sigma(C) = \sum_{u \in C} \sigma(u, C)/|C|$. Higher community safeness means finding the correct set of updates. Based on the community safeness, the Ds algorithm is proposed to deceive a detection algorithm by changing (adding/deleting) a certain number of links of the target community. Let $\Delta(C) = \sigma(C') - \sigma(C)$ be the safeness gain, where C' is the community after one or more link updates. The Ds is a greedy algorithm which selects the edge update that gives the highest $\Delta(C)$ at each step. The Ds algorithm only considers inter-community edge additions and intra-community edge deletions. Additionally, in the study (*13*), the Dm algorithm that is based on the modularity loss is presented for community deception. However, it needs the knowledge of the community structure. Later, Fionda and Pirro (*110*) offer the SECRETORUM algorithm, in which node safeness is re-defined to work with weighted networks. Further, the same author group explores community deception for directed and/or weighted graphs in the studies (*111–113*).

In the EPA method (*14*), for target community and global attacks, the fitness function is calculated using entropy and degree change. Entropy is employed to assess the attack effect using the community detection results before and after the attack. Therefore, the EPA method requires the knowledge of the community structure discovered by a particular detection algorithm. The fitness function for a global attack is designed as follows:

$$f = \Psi(d') \times (Ent_A/\log_2 |\widehat{C}| + Ent_B/\log_2 |\overline{C}|), \tag{3.2}$$

Ent_A and Ent_B are the entropy for new communities and original communities, respectively, m is the confusion matrix of two community distributions, m_{ij} is the number of common nodes between the original community C_i and the new community C'_j, n is the number of nodes in the graph, and

$$Ent_A = -\sum_{i=1}^{|\overline{C}|} \sum_{j=1}^{|\widehat{C}|} \frac{M_i}{n} \left(\frac{m_{ij}}{M_i} \log_2 \frac{m_{ij}}{M_i} \right). \tag{3.3}$$

The maximum value of Ent_A (resp., Ent_B) is obtained if the values of each row (resp., column) are equal. For a target community attack, the fitness function given in Equation (3.2) can be adapted to limit the search space (*14*).

Chen et al. (*114*) give a new definition for community safeness that considers the

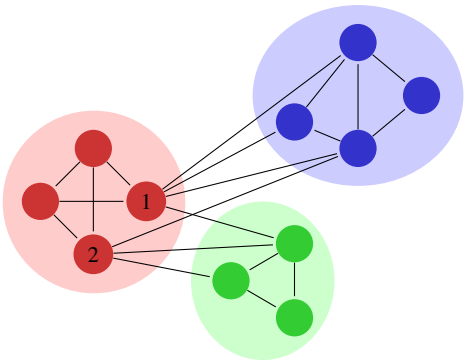
total shortest path for every pair of nodes in the target community C. For this, the degree of dispersion of the community C is defined as $\rho(C) = \sum_u \sum_v sp(u, v)$, where $u, v \in C, u \neq v$ and $sp(u, v)$ is the length of the shortest path between node u and node v. The higher $\rho(C)$ is, the better the community hiding. The $\rho(C)$ is normalized to maintain it in the range $[0, 1]$, that is, $\psi(C) = (\rho(C) - \rho(C)_{min})/(\rho(C)_{max} - \rho(C)_{min})$. Then, the community safeness is defined as $\sigma(C) = \frac{1}{2}\psi(C) + \frac{1}{2}\varphi(C)$, where $\varphi(C) = \sum_{u \in C} \frac{|\tilde{E}(u, C)|}{deg(u)}$. The Ds algorithm is improved, and the Hs algorithm, which is based on the new safeness definition, is proposed.

Mittal et al. (*77*) use permanence for community deception. Permanence is a node-based metric which quantifies the belongingness of a node v to its community C (*115*):

$$Perm(v) = \frac{I(v)}{E_{max}(v)} \times \frac{1}{deg(v)} - (1 - c_{in}(v)), \tag{3.4}$$

which is based on three factors: 1) internal pull $I(v)$, the internal connections of v inside its own community; 2) maximum external pull $E_{max}(v)$, the maximum connections of v to its neighboring communities; and 3) v's internal clustering coefficient $c_{in}(v)$, the fraction of the actual and possible number of links among the internal neighbors of v. The permanence of a node increases if its internal pull is greater than its external pull or its internal neighbors are densely connected. Figure 3.4 illustrates a toy example of the calculation of the permanence of two nodes. The permanence of a network G is calculated as $Perm(G) = \sum_{u \in V} Perm(v)/|V|$. Mittal et al. propose NEURAL, a permanence-based deception method that aims to reduce the network permanence to hide a target community C. NEURAL is a greedy strategy that maximizes the permanence loss $P_l = Perm(G) - Perm(G')$ by choosing the edge update giving the highest P_l at every iteration. NEURAL also allows only intra-community edge deletions and inter-community edge additions. The work (*116*) then adapts the permanence formula to operate on weighted networks and offers the PERMDEC algorithm using the permanence loss.

The Matthew effect caused by traditional quality function (i.e., modularity)-based methods is identified, where earlier edge perturbation influences the placement of subsequent perturbations at the community level (*81*). To address this issue, a probabilistic framework, ProHiCo, is designed to hide a set of target communities. The core concept behind it is to initially allocate the resource of perturbations randomly and sequentially, followed by selecting the suitable edges for perturbation through likelihood minimization. By integrating



Node v	1	2
$deg(v)$	6	5
$I(v)$	2	2
$E_{max}(v)$	3	2
$c_{in}(v)$	1	1
$Perm(v)$	0.11	0.2

Figure 3.4. A toy example illustrating the calculation of permanence of two nodes.

the stochastic block model and its degree-corrected version into the ProHiCo framework, two scalable algorithms, SBM and DCSBM, employing sampling and pruning techniques, are proposed. Then, the scalable ComDeceptor method (*117*), leveraging the Laplacian spectrum, also hides an arbitrary set of target communities. It first allocates the resources of perturbations fairly, giving either a pair of communities for edge addition or a community for edge deletion in each round. The method performs inter-community edge additions by maximizing the second smallest Laplacian eigenvalue while performing intra-community edge deletions by minimizing the largest Laplacian eigenvalue. It incorporates heuristics for approximately solving them.

A node-centric approach (*118*) is introduced for community deception by allowing node updates, where each node operation (node addition or node deletion) results in several edge updates. It relies on node safeness defined in Equation (3.1). A greedy algorithm, nSAF, is proposed that selects the node operation (between a node addition candidate and a node deletion candidate) that yields the best safeness gain for each step, along with the respective edges. For deletion, the node with the least safeness is chosen among the target community nodes, and its edges are determined as the edges to be deleted. For the addition of a new node (natural or bot), the edges to be added are determined as follows: first, only one edge is established with a node in the target community to form a single component, and then, the remaining edges are established with nodes outside the target community. Another greedy algorithm with a node-centric approach, nDec (*119*), is based on modularity and considers three node operations (addition, deletion, and moving). Node addition is essential when new members join the network. While moving nodes between communities is generally preferable, node deletion may be necessary in cases where the complete concealment of the target community is critical. A deleted node can be re-added

with carefully chosen edge additions.

Chang et al. (*120*) offer three genetic algorithms for hiding a target community using escape score, dispersion score (deception score H in (*13*)), and hiding score as the fitness functions. The escape score quantifies the number of nodes in the target community that are concealed. The hiding score formula is obtained from the first two scores.

A swarm intelligence-based method, SCP, is introduced for target community attacks (*121*). The edge pool is initialized according to permanence, as in (*11*) (explained in Section 3.2.3), but only the edges related to the target community are considered. The search space is tailored for each particle. A self-adaptive mechanism is implemented to balance global and local search resources. The fitness function is derived from the structural entropy given in Equation (3.6).

Table 3.2 summarizes and compares target community attack algorithms. For each study, reference of the study, target community attack algorithm name(s), update operation, whether intra-/inter-commmunity edges are used or not, knowledge needed (if any prior information required), and measures used to compare the attack algorithms with the algorithms in the comparison column are specified in the table. The table indicates that manipulation of the network structure exclusively through the addition of edges is achieved only in the study (*76*). The other algorithms modify the network by both adding and removing edges. Additionally, nSAF and nDec algorithms also perform node updates. Notably, edge rewiring is implemented only in the EPA and CH-SNMF algorithms. Rewiring is not applied in greedy optimization-based methods (i.e., Ds, Dm, SECRETORUM, Hs, NEURAL, nSAF, and nDec). Except for EPA, nSAF, and nDec, all of them perform inter-community link addition and intra-community link deletion since the target community to be concealed is known and other communities can be treated as external communities. Some algorithms may need additional prior information to hide a target community. For example, information about neighboring communities is needed to calculate the permanence value in the NEURAL algorithm. Certain algorithms even demand a community structure extracted from the network by a community detection algorithm, implying the necessity of knowing the entire network. This requirement may potentially restrict the practical applicability of these methods in real-world scenarios. When evaluation measures are analyzed, it is observed that the community deception score H, which captures all three desiderata of effective community hiding, and NMI are

Table 3.2. Summary of target community attacks.

Ref.	Community Hiding	Update	Intra/ Inter	Knowledge	Measure	Comparison
(*76*)	Naga.	EAdd	✓	C, Vertex centralities	Miss ratio	No
(*12*)	DICE	EDel, EAdd	✓	C	M	No
(*13*)	Ds, Dm	EDel, EAdd	✓	C for Ds, CS for Dm	H, NMI	DICE
(*14*)	EPA	EDel, EAdd (Rewire)	✗	G, A part of CS	Fitness, H	DICE, Ds
(*110*)	SECRETORUM	EDel, EAdd	✓	C	H, NMI	Rand, DICE, Ds, NEURAL
(*114*)	Hs	EDel, EAdd	✓	C	H	Ds, Dm
(*77*)	NEURAL	EDel, EAdd	✓	Node info for a subset of nodes	NMI, MNMI, CommS, CommU	Rand, Naga., DICE, Ds
(*81*)	ProHiCo (SBM, DCSBM)	EDel, EAdd	✓	G, CS	J, R, P, NMI	Ds, REM
(*117*)	ComDeceptor	EDel, EAdd	✓	G, CS	J, R, P, NMI	Ds, REM, DCSBM
(*74*)	DRL-Agent	EDel, EAdd	✓	G, CS	H, NMI	Ds, Dm
(*118*)	nSAF	EDel, EAdd, NDel, NAdd	✗	C	H, NMI	Rand, DICE, Ds, Dm, NEURAL
(*119*)	nDec	EDel, EAdd, NDel, NAdd, NMov	✗	G, CS	H, MNMI, NMI	Rand, DICE, Ds, Dm, NEURAL
(*75*)	CH-SNMF	EDel, EAdd (Rewire)	✓	G, $\|CS\|$	H, M	DICE, Ds
(*120*)	CEHA, CDHA, CCHA	EDel, EAdd	✓	G, CS	NMI, Q, M	Rand, DICE, Ds, NEURAL
(*121*)	SCP	EDel, EAdd	✓	G, CS	NMI, VI, SJD, Local	Rand, DICE, Ds, NEURAL, MOD

predominantly preferred. Following them, the measure M is preferred. In the comparative analysis of the attack algorithms, the Ds algorithm emerges as the most frequently utilized, indicating its widespread acceptance and effectiveness in the context of target community attacks.

3.2.3. Global Attack

In a global attack, all communities within the network are regarded as sensitive. This attack focuses on modifying the least number of connections to significantly alter the overall community structure, thereby ensuring privacy.

Definition 10 (Global Attack) *Let $G = (V, E)$ be a graph. f is a community detection algorithm that discovers the community structure $\overline{C} = \{C_1, \ldots, C_K\}$ on G. The global attack problem aims to maximize the change in the community structure by allowing a certain number (β) of edge modifications (addition or deletion) in the network, where β is the budget. After the attack, the adversarial graph G' is obtained. The community structure discovered from G' is $\widehat{C} = \{C'_1, \ldots, C'_L\}$ through the community detection algorithm f. The community structure $\widehat{C}$ is as different as possible from the original community structure $\overline{C}$.*

Figure 3.5 presents a visualization of a global attack on the Karate network (*108*). The implementation of the Louvain algorithm divides the nodes in the original network into four communities, each represented by a different shape to indicate the membership of the individuals. Performing a global attack, consisting of four strategically selected edge rewirings, produces the adversarial network; the resulting community structure on it by the same detection algorithm is illustrated in Figure 3.5b (*8*), revealing the change in the number of communities and exhibiting disorganization at a broader scale by modifications across the network.

The attack strategies for global attacks are designed to disrupt the community structure. Figure 3.6 shows an ideal case for community obfuscation.

Chen et al. (*8*) tackle the problem of global community structure deception and develop strategies to attack community detection algorithms by rewiring the minimal number of links. Two heuristic attack strategies, community detection attack (CDA) and degree-based attack (DBA), as baselines, and a genetic algorithm-based strategy, Q-Attack,

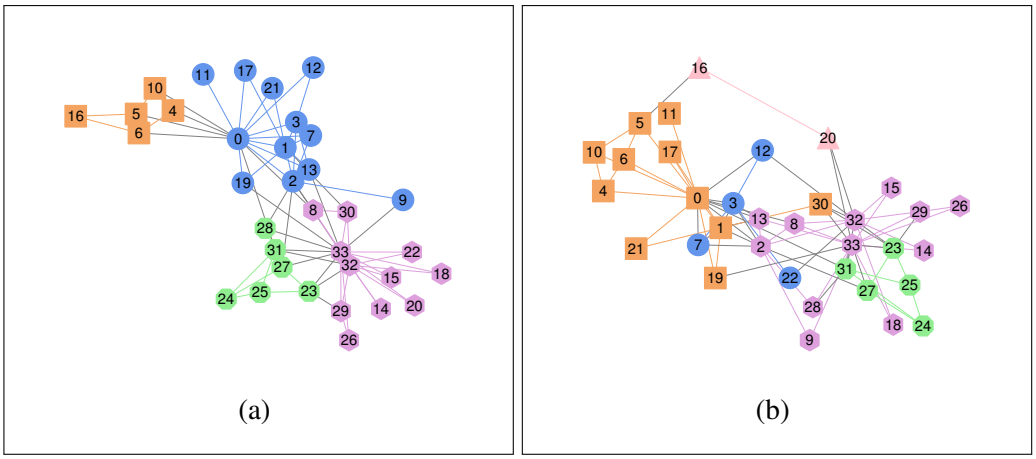


Figure 3.5. Illustration of a global attack. (a) Community structure of the original graph. (b) Community structure after applying a global attack.

are given to fool the detection algorithms. These strategies rely on rewiring, that is, adding an edge to a node while deleting one from it. The CDA strategy uses knowledge of the community structure found by a specific detection algorithm. Since there is no target community, it randomly selects a certain number of nodes from the network. In every iteration, for a chosen node, an existent intracommunity edge is deleted and a non-existent inter-community edge is added. The DBA heuristic strategy is also similar to CDA but chooses the nodes of larger degree from the network.

The genetic algorithm is used to search for the optimal set of rewiring links (*8*). The Q-Attack algorithm is mainly composed of three parts: encoding, fitness function, and design of genetic operators. For encoding, an attack is represented by a chromosome and a rewiring (a deleted edge and an added edge) is represented by a gene. An example individual consisting of four rewirings is demonstrated in Table 3.3. The fitness function

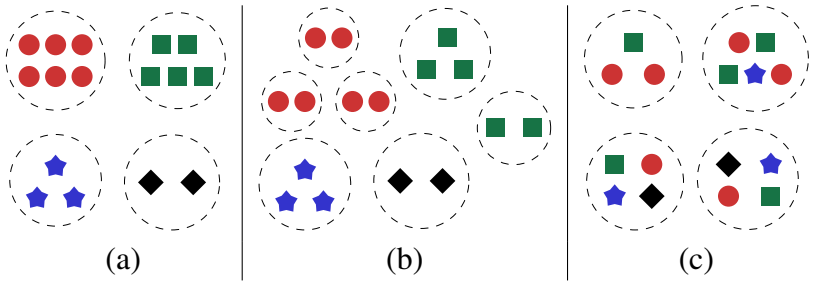


Figure 3.6. The objective of global community obfuscation is demonstrated: (a) original community structure; (b) community dispersion occurring by dividing the original community; (c) optimal case of community obfuscation, with nodes of each community spread among various communities (*11*).

used is $f = e^{-Q}$, which indicates that individuals with lower modularity will have larger fitness. The probability of selecting an individual is proportional to its fitness, represented as $p_i = f(i)/\sum_{j=1}^{n} f(j)$. A single-point crossover between two individuals is adopted with the probability p_c, and to promote the diversity, three types of mutation operators, such as link deletion, link addition, and link reconnection, are proposed with probability p_m.

Table 3.3. An individual representation in the Q-Attack (*8*).

(19, 38)	(29, 21)	(25, 33)	(20, 15)
(19, 13)	(29, 14)	(25, 28)	(20, 5)

In the study (*80*), structural entropy for a graph $G = (V, E)$ is expressed as follows:

$$\mathcal{H}(G) = -\sum_{i=1}^{|V|} \frac{d(i)}{2|E|} \log_2 \frac{d(i)}{2|E|}, \tag{3.5}$$

where $d(i)$ is the degree of node i. Then, the structural entropy for the graph with respect to the community structure $\overline{C} = \{C_1, C_2, \ldots, C_k\}$ is defined, in which $v(j)$ is the sum of degrees in community C_j and $g(j)$ is the number of external edges from community C_j.

$$\mathcal{H}_{\overline{C}}(G) = -\sum_{j=1}^{k} \left[\frac{v(j)}{2|E|} \left(-\sum_{i \in C_j} \frac{d(i)}{v(j)} \log_2 \frac{d(i)}{v(j)} \right) - \frac{g(j)}{2|E|} \log_2 \frac{v(j)}{2|E|} \right]. \tag{3.6}$$

To disrupt the community structure, their REM method minimizes $\rho = (\mathcal{H}(G) - \mathcal{H}_{\overline{C}}(G))/\mathcal{H}(G)$. It only performs edge additions between communities by considering nodes with a low degree.

The study (*122*) first examines the effects of attacks with different edge-importance measures, like edge degree (Deg), betweenness centrality (Bet), and closeness centrality (Clo). Edges are removed based on these measures. It then addresses building robust community structures that can tolerate such failures.

Network embedding algorithms map the nodes of a network into the vectors in a Euclidean space. These vectors then can be used by downstream network tasks, such as node classification, community detection, and link prediction. Yu et al. (*123*) focus on attacking the network embedding process. An attack method, namely EDA, is proposed for disturbing the distances between the embedding vectors through minimal changes of the network structure. For embedding, the DeepWalk method is used. To obtain the optimal set of modified links, a genetic algorithm is used. The EDA method iteratively calls DeepWalk

and the genetic algorithm and returns the adversarial network at the end. This attack method disturbs the global network structure. Moreover, to observe the effectiveness of the attack on community detection, nodes are converted into vectors using DeepWalk and the embedding vectors are clustered via the K-means algorithm. As an unsupervised attack, EDA does not need any knowledge of communities.

Persistence, which includes the same three factors and is too similar to permanence, is defined in the study (*82*). However, persistence is used for global attacks. Two attack strategies, maximization of persistence loss (MPL) and a rewiring strategy, are offered. The MPL strategy aims to maximize the persistence loss of the network. A node is moved to a neighboring community if its persistence score decreases. All the nodes in the network are checked. In MPL, the structure is not altered. A rewiring strategy can be applied after MPL to change the structure of the network. For every shifted node, an edge from the old community is deleted and a new edge is added to a node in the new community.

Modularity vitality reflects the role of a node (i.e., bridge or hub) within a given community structure (*124*). This measure can be utilized to execute global attacks, with a greedy approach by removing community hub nodes—those with high modularity vitality. Even an approximation is offered with the calculation of the values only once. Experiments on a network reveal that a specific level of attack effect can be achieved by removing a few nodes or by removing a large number of edges. The suggestion is that users seeking to safeguard their community identity break their connections with community leaders.

Graph neural networks (GNNs) have been widely preferred in graph-related tasks. Liu et al. (*83*) apply a graph auto-encoder (*125*) to handle the community hiding problem and propose a method called GCH. A graph auto-encoder is used to reconstruct the probability adjacency matrix. The GCH method takes the community structure discovered by a detection algorithm to generate the adversarial network. The original network is perturbed by deleting the edges with the highest probability within communities and adding the edges with the lowest probability between communities. In this manner, the global community structure is disturbed, which leads the structure to be very different.

Liu et al. (*84*) propose a community hiding algorithm based on genetic algorithms using normalized mutual information (NMI), called CGN. This algorithm realizes global community structure hiding. It mainly consists of four parts: creating a gene pool with prior information, encoding, fitness function, and genetic operators. A gene pool is established

according to preliminary information about the community structure of the original graph found by a community detection algorithm. That means rather than considering all existent graph edges for deletion and all non-existent graph edges for addition, the gene pool includes existent edges within the communities (intra-community edges) for deletion and non-existent edges between the communities (inter-community edges) for addition. In encoding, a chromosome corresponds to an attack and an edge randomly chosen from the gene pool corresponds to a gene. The fitness function is designed as $f = 1 - NMI$, which indicates the chromosome with a lower NMI value will have larger fitness and produce a better attack. The benefit of employing NMI as a fitness function is that the minimum value of NMI shows that the community structure of the network has changed significantly after the attack. The roulette wheel selection is used. The chromosomes in a population are mapped to a wheel, where those with higher fitness values have a higher probability of being selected. A single-point crossover is applied between two chromosomes with the probability p_c, and mutation is performed with probability p_m.

Zhao et al. (*11*) develop SAEP (a self-adaptive evolutionary deception) framework. It offers a permanence-based edge pool initialization mechanism. The permanence of a node is a measure of loyalty to the community it belongs to. A permanence value is in $(-1, 1]$. A value close to 1 means a loyal node, while close to -1 means a tendency to leave. For the edge deletion, intra-community edges are sorted by the product of their nodes' permanence in increasing order. This implies that in a community, the nodes with low loyalty are more likely to leave because they lose the attraction from the ones with high loyalty. For the edge addition, it focuses on the non-existent inter-community edges that are sorted by the product of the nodes' permanence in increasing order. It means that a node with low permanence in community A receives an invitation from a node loyal to an external community B. Further, a penalty for adding edges to the node from unrelated communities is defined by multiplying the result by 0.5. Then, edges are sampled from the pool with a probability $prob(i) = \frac{x_i^{-\theta}}{\sum_{k=1}^{|P|} x_i^{-\theta}}$, where x_i is the ith element in the sorted pool, $|P|$ is the pool size, and θ is empirically assigned to 0.3.

A fitness function is proposed to capture local and global community change (*11*). It consists of multiple components ($f = f_1 * f_2 * f_3$). To capture local change, the assumption is that if an edge is modified (added/deleted), its two nodes should be affected first. The aim is to observe if the nodes of the modified edge join the other communities.

A set of nodes $\Delta(V')$ related to the modified edges $\Delta(E')$, which includes nodes whose community is not the same as that of its most loyal neighbor after the attack, are identified. f_1 is designed as $f_1 = \sum_{v \in \Delta(V')} \log_2 d(v)$. To capture global change, two functions are defined based on the confusion matrix. f_2 is Ent_A defined in the study (*14*). This will force nodes to leave their initial community. f_1 cannot capture local obfuscation if the entire community (the affected node and its most loyal neighbor) combines with another community after the attack. To solve the issues, a variant of the confusion matrix m', is introduced as $m'_{ij} = \frac{m_{ij}}{|C'_j|} * \log_2 |C'_j|$. The first part $\frac{m_{ij}}{|C'_j|}$ is the proportion of nodes of the original community C_i that form the new community C'_j. The log part is to prevent a situation like a community with one node. f_3 is defined as $f_3 = e^{-max(m')}$ and tries to make the maximum value as small as possible.

In the SAEP (*11*), each chromosome is assigned a strength vector to evaluate the strength or weakness of its genes. The vector for the jth chromosome is $W^j = [W^j_1, \ldots, W^j_\beta]$. An edge distance is introduced to ensure that the edges in the solution are close to each other. The penalty Q of a modified edge is $Q(e) = \Sigma^k_{i=1}\sigma^i$, where σ is set in $[0, 1]$ and k indicates there are no modified edges at k-hop distance. The strength of the ith edge (gene) in its chromosome is updated as $W_i = W_i * \frac{1}{1 + exp(Q(e))}$ if $k > 1$. Moreover, adaptive crossover and mutation operations are designed according to the strength vectors. To exchange only the inferior genes, a uniform crossover operation is used; exchanging of two genes is performed if a random number is greater than the maximum of the W^m_i and W^f_i that represent the ith gene of the mother chromosome and father chromosome, respectively. Then, the weights of the changed genes are updated according to the fitness gain. In mutation, the weaker a gene is, the more likely it is to mutate. Also, the weight of the changed gene is updated.

A heuristic algorithm, degree first deception (DFP), is proposed (*11*). It assumes that nodes with a large degree have a greater impact on the community structure. Therefore, it prioritizes the deletion of intra-community edges between large-degree and small-degree nodes and the addition of inter-community edges with large-degree nodes. That is, it sorts the edges using the following rule:

$$S(u,v) = \begin{cases} \dfrac{min(d_u, d_v)}{max(d_u, d_v)}, & (u,v) \in E, C_u = C_v \\ d_u + d_v, & (u,v) \in \overline{E}, C_u \neq C_v. \end{cases} \tag{3.7}$$

where E and $\overline{E}$ are all edges and non-existent edges in the graph, d_x is the degree of node x, and C_x is the community that x belongs to. The study (*126*) later proposes a modified version called DFP-R, which uses the same metric for the addition and removal of edges, but the created edge is chosen to have a vertex in common with the removed edge.

A coevolutionary method (*127*), called CoeCo, to obfuscate the community structure is offered to also apply on large-scale datasets. It divides the graph into multiple similar-sized subgraphs, and each is optimized separately. In the method, to reduce the search space, initialization is performed. Edges are sorted according to permanence multiplication as in the SAEP (*11*) for addition and motif weights (the number of triangles containing the edge) for deletion. Two fitness functions are adapted, namely the fitness function for the ith subgraph $f_i = \mathcal{H}(subG_i)$ defined in Equation (3.6) (*80*) and the fitness function for the global graph $f_{glob} = f_1 * f_2 * f_3$ introduced in (*11*). In the coevolution part, they support each other to identify the optimal set of edges.

Another genetic algorithm-based method for global attack is EPCG (*128*), which employs two different fitness functions (based on NMI and the difference between an individual and the best one to preserve the population diversity). For better concealment of attack, the number of edge additions and deletions is kept equal. It adopts a co-evolution mechanism with two elite populations to improve evolution.

The study by (*129*) combines a graph auto-encoder (GAE) and genetic algorithm to deceive community detection algorithms. Initially, small subnetworks are sampled and reconstructed using the GAE with an added community hiding constraint in the loss function. Subsequently, a genetic algorithm is applied to improve the hiding effects through the network, treating each reconstructed subnetwork as an individual. The genetic algorithm fitness function and operators utilized follow those in Chen et al. (*8*), with elitism retaining the top 15% of individuals.

A heuristic approach based on local structures, namely LSHA, is proposed (*126*). It does not depend on the community structure produced by a particular community detection algorithm. Instead, local structures are detected using the local information of nodes. A community can contain multiple local structures. LSHA chooses two local structures (a low degree one and a high degree one) using the modularity concept for attack. Then, to choose the edge in the low-degree structure, an edge vulnerability metric is proposed, which measures the significance of the edge in preserving the structure. This metric can be

formally defined as follows:

$$Vul(e) = \frac{|N(v_i)| - |N(v_i) \cap N_{LS}(v_i)| + |N(v_j)| - |N(v_j) \cap N_{LS}(v_j)|}{|N(v_i)| + |N(v_j)|}, \tag{3.8}$$

where e is an edge (v_i, v_j), $N(v_x)$ is the set of neighbors of v_x, and $N_{LS}(v_x)$ is the set of nodes within the local structure to which v_x belongs. For deletion, the edge with the high vulnerability is chosen, and its lower degree node is chosen for rewiring around it. For addition, the other node is selected from the high degree structure according to the node entropy metric that quantifies the perplexity level of a node for the partition. For a node v, it is formally defined as

$$Ent(v) = -\sum_{i}^{M} \frac{|LS_i \cap N(v)|}{|N(v)|} \log\left(\frac{|LS_i \cap N(v)|}{|N(v)|}\right), \tag{3.9}$$

where LS_i is the local structure that has connections with node v and M is the number of such structures. The entropy increase is calculated for each node, and the node with the most increase is selected to connect with.

Table 3.4 summarizes and compares global attack algorithms. For each study, reference of the study, global attack algorithm name(s), update operation, whether intra-/inter-commmunity edges are used or not, knowledge needed (if any prior information is required), and measures used to compare the attack algorithms with the algorithms in the comparison column are specified in the table. The table shows that in the REM algorithm, modification is performed solely through the addition of edges, while other algorithms, except for the studies (*122*) and (*124*), employ both edge addition and removal to alter the network structure. Rewiring is applied in heuristic approaches, such as DBA, CDA, LSHA, DFP-R, and Custom Rewiring (*82*). Although some genetic algorithm-based methods (Q-Attack and EPA) incorporate rewiring, some of them (EDA, CGN, SAEP, CoeCo, EPCG, and GAE+Genetic alg (*129*)) do not, with the assumption that the genetic algorithm might not achieve its full potential efficiency (*84*). Heuristic methods, with the exception of (*122*) and (*124*), leverage intra-community and inter-community information (intra-community edge deletion and inter-community edge addition) to guide their structural adjustments. While some genetic algorithm-based methods (Q-Attack, EPA, EDA, EPCG, and GAE+Genetic alg) do not utilize intra/inter-community information, recent methods like CGN, SAEP, and CoeCo have integrated it to enhance their effectiveness.

Moreover, most global attack methods rely on a community structure, though some methods (EDA, GAE+Genetic alg, LSHA, and the methods in the study (*122*)) are designed to function without prior knowledge of the communities, broadening their applicability in various scenarios. When evaluating the performance of the attack algorithms, NMI emerges as the most frequently used measure, with ARI being the second most commonly employed measure.

Table 3.4. Summary of global attacks.

Ref.	Community Hiding	Update	Intra/ Inter	Knowledge	Measure	Comparison
(*8*)	DBA, CDA	EDel, EAdd (Rewire)	✓	G, CS	Q, NMI	Rand-R
	Q-Attack		✗	G, CS, Q		
(*80*)	REM	EAdd	✓	G, CS	J, NMI, R	MOM, Rand-Add
(*14*)	EPA	EDel, EAdd (Rewire)	✗	G, CS	NMI, ARI	Q-Attack, EPA with H, two heuristics
(*122*)	Deg, Bet, Clo	EDel	✗	G	NMI	Rand-Del
(*123*)	EDA	EDel, EAdd	✗	G	NMI	Rand, DICE, RLS, DBA
(*82*)	MPL, Custom Rewiring	EDel, EAdd (Rewire)	✓	G, CS	Node-centric, NMI, CCs	Ds
(*124*)	Modularity Vitality	EDel, NDel	✗	G, CS	Modularity Minimization	No
(*83*)	GCH	EDel, EAdd	✓	G, CS	NMI, Attack Effic.	Rand, DICE, Ds
(*84*)	CGN	EDel, EAdd	✓	G, CS	Q, NMI, BN (β * NMI)	Rand, DICE, Ds, Q-Attack, NEURAL
(*11*)	SAEP, DFP	EDel, EAdd	✓	G, CS	NMI, VI, SJD	Rand, REM, Q-Attack, DFP
(*127*)	CoeCo	EDel, EAdd	✓	G, CS	NMI, ARI	Rand, REM, Q-Attack, DFP
(*128*)	EPCG	EDel, EAdd $\lvert E^+\rvert = \lvert E^-\rvert$	✗	G, CS	NMI, ARI, Purity	Rand, CDA, Q-Attack
(*129*)	GAE + Genetic alg	EDel, EAdd	✗	G	NMI, ARI	Rand, Barabasi Albert, CDA, Q-Attack
(*126*)	LSHA	EDel, EAdd (Rewire)	✓	G	NMI, ARI	Rand-R, CDA, DBA, DFP-R
(*75*)	CH-SNMF	EDel, EAdd (Rewire)	✓	G, $\lvert CS\rvert$	NMI, ARI	Rand, Q-Attack

CHAPTER 4

PROPOSED METHODS FOR KNOWLEDGE HIDING ON GRAPH DATA

This chapter presents the algorithms developed in this thesis to tackle the problems associated with graph hiding. It begins by introducing a set of algorithms specifically designed to address the subgraph hiding problem, including the Edge Deletion-based Heuristic (EDH) (*15*), EdgeDegree, Matchings & EdgeDegree, EdgeEntropy, and Matchings & EdgeEntropy. These algorithms employ strategic modifications to the graph database, effectively concealing sensitive subgraph patterns while maintaining the utility of the graph data for other analyses. Following this, the chapter moves on to the proposed global community hiding algorithm, referred to as 4-clique Community Hiding (4clqCH). This algorithm is designed to disrupt the detection of overlapping cohesive communities by targeting the cliques appearing in the graph, hence reducing the effectiveness of the community detection algorithms. Together, these algorithms offer different solutions to safeguard privacy in graph-based systems through graph manipulation techniques.

4.1. Subgraph Hiding Methods

In this thesis, different subgraph hiding algorithms are proposed; EDH, EdgeDegree, Matchings & EdgeDegree, EdgeEntropy and Matchings & EdgeEntropy. These algorithms hide the sensitive subgraphs by decreasing their frequencies under the predefined disclosure threshold through deleting the victim edges from some sensitive graphs. To simplify the complexity of the solution, various heuristics are used in the design of the algorithms, which are for (i) determining the graphs for sanitization and (ii) determining the edges to be removed from the chosen graphs.

The edge deletion-based subgraph hiding algorithms are expressed on an example graph given in Figure 4.1. On the graph, the matches of a sensitive pattern are shown with red and green colors. There are two matches; one of them includes the edges with the green and both colors, and the other one with the red and both colors.

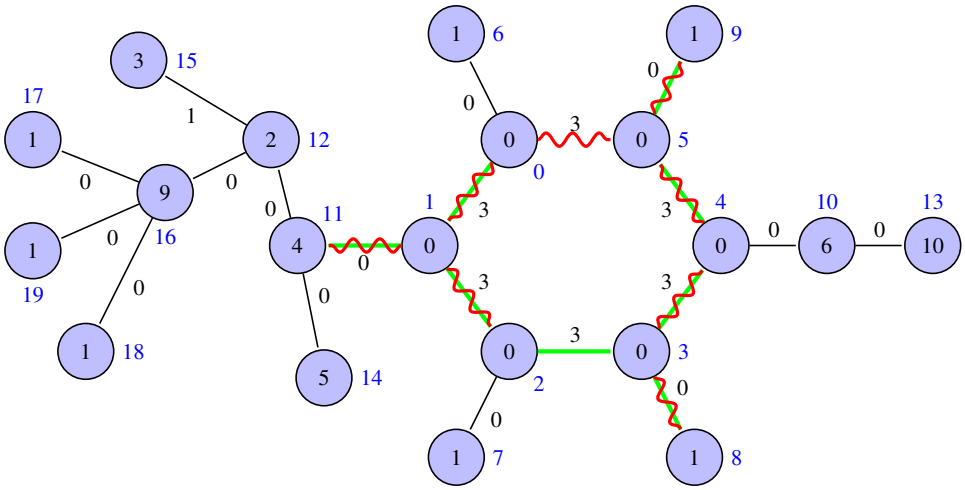


Figure 4.1. An example graph with the two matches of a sensitive pattern. Each match is shown with a color.

4.1.1. Edge Deletion-Based Heuristic (EDH) Algorithm

The Edge Deletion-based Heuristic (EDH) algorithm sanitizes the transactional graph database to hide the set of sensitive subgraph patterns by decreasing the frequency of every sensitive pattern below the predefined disclosure threshold. If all matches of a sensitive pattern are deleted from a sensitive graph in the database, its frequency decreases by one. That means, a sensitive pattern is hidden by removing sufficient amount of edges from sufficient amount of sensitive graphs. The EDH algorithm hides all the sensitive patterns without hiding failure. The main idea of the EDH algorithm is to remove the most common edge in all matches of a sensitive pattern within a selected sensitive graph. Removing the most frequent edge from the graph will eliminate most of matches of the sensitive pattern at the same time. The EDH algorithm consists of five main steps as follows:

Step1: Calculating the relevance of the graphs and and the sensitive patterns. To determine the relevance of each graph to the set of the sensitive patterns, the number of matches of all sensitive patterns in each graph is found. Also, the relevance of each sensitive pattern to the graph database is determined by calculating the number of matches it has in all the graphs.

Step2: Sorting the graphs. The graphs are sorted in non-descending order based on their relevance numbers calculated in Step 1.

Step3: Sorting the sensitive patterns. The sensitive patterns are sorted in non-

descending order based on their relevance numbers calculated in Step 1.

Step4: Calculating the number of sensitive graphs for sanitization. Based on the given disclosure threshold ψ, the number of sensitive graphs required to be sanitized to conceal each sensitive pattern P is calculated by $freq(P) - \psi + 1$. The disclosure threshold is a direct measure, rather than a percentage in order to balance the disclosure and privacy, which ensures that no sensitive pattern is revealed.

Step5: Sanitizing a sensitive graph (called local sanitization). This step involves sanitizing a sensitive graph by removing all the matches of a given sensitive pattern. Local sanitization in the EDH algorithm is achieved by deleting the chosen edges from the graph.

Algorithm 1 EDH Algorithm.

Input: Database D, Sensitive patterns SP, Disclosure threshold ψ
Output: Modified Database D'.

```
1:  for each sensitive pattern P ∈ SP do          ▷ Set P.matches and G.matches to 0.
2:      pMatchNum ← 0
3:      for each G ∈ D do
4:          matchNum ← |M(G, P)| where M(G, P) keeps the matches
5:          pMatchNum ← pMatchNum + matchNum
6:          G.matches ← G.matches + matchNum
7:      end for
8:      P.matches ← pMatchNum
9:  end for
10: Sort D by G.matches in non-descending order
11: Sort SP by P.matches in non-descending order
12: for each sensitive pattern P ∈ SP do
13:     NumGraphsForSanitization ← freq(P) − ψ + 1
14:     for i ← 1 to NumGraphsForSanitization do
15:         Get next sensitive graph G ∈ D that includes P
16:         LocalSanitizationOfEDH(G, P)
17:     end for
18: end for
19: D′ ← D
```

The EDH algoritm is depicted in Algorithm 1. Step 1 calculates the relevance of the graphs and sensitive patterns outlined in lines 1 to 9. The $matchNum$ represents the number of matches of a sensitive pattern appearing in a graph in the database. The total $pMatchNum$ value for each sensitive pattern across all graphs in the dataset is calculated and stored in $P.matches$. For each graph, the $G.matches$, which is the total number of matches across all sensitive patterns, is calculated. In step 2, the graphs are sorted in non-descending order based on the values of $G.matches$ at line 10. In step 3, sorting the sensitive

patterns in non-descending order is performed at line 11. After that, the followings are done for each sensitive pattern $P \in SP$: in step 4, $NumGraphsForSanitization$ is determined as the number of graphs to sanitize at line 13, and $NumGraphsForSanitization$ sensitive graphs (containing the sensitive pattern) are sanitized by removing all the matches of the pattern. Therefore, in Step 5 to sanitize a chosen sensitive graph for a sensitive pattern locally, LocalSanitizationOfEDH (Algorithm 2) is called:

1. All matches of the sensitive pattern in the graph are identified (M). Each match in M includes the pairs of the vertex id in the graph and its associated vertex id in the pattern. The keys in each match are exchanged with their values, and M is sorted based on the values.
2. All the graph edges in M are identified (lines 5-11).
3. The most frequent graph edge appeared in M is found. If multiple graph edges have the same frequency in M, the first one listed in E is chosen.
4. The chosen graph edge is deleted from the graph.
5. The matches that include the graph edge are removed from the M, and the process is repeated starting from 2 until no matches remain.

Algorithm 2 LocalSanitizationOfEDH(G, P).

```
1:  M ← Determine the matches M(G, P)
2:  Replace values with keys in each match of M
3:  Sort M acc to values
4:  while M is not empty do
5:      E ← list
6:      for each match m ∈ M do
7:          for (u, v) ∈ P.edges do
8:              (gu, gv) ← (m[u], m[v])
9:              Insert (gu, gv) into E
10:         end for
11:     end for
12:     e ← The most frequent graph edge in E
13:     Delete edge e from graph G
14:     Remove M_e from M where M_e keeps the matches containing e
15: end while
```

Let us use the EDH algorithm for local sanitization of the sensitive graph given in Figure 3.1a to compare it with the masking-based algorithm. As explained in the algorithm, all matches of the sensitive pattern displayed in Figure 2.1b are deleted. For this, the EDH identifies all the matches present in the graph. The graph edges in the matches are

determined. The EDH identifies the edge $(2, 3)$ as the most common graph edge in the matches and deletes it. The matches containing the edge $(2, 3)$ are deleted from the M. Then, only one match remains and the process continues with this match. It finds $(0, 5)$ as one of the most common edges in the match, and deletes it. Figure 4.2 shows the graph after sanitizing with the EDH algorithm. This algorithm does not produce new fake patterns.

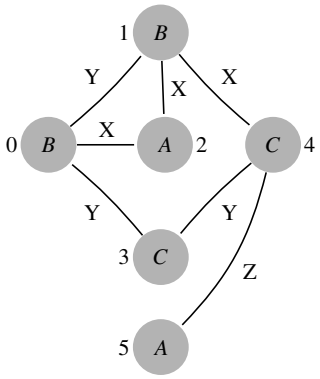


Figure 4.2. The graph after sanitizing with the EDH algorithm.

4.1.2. EdgeDegree Algorithm

As the EDH algorithm, the EdgeDegree algorithm is proposed for subgraph hiding. The EdgeDegree algorithm also hides the given sensitive subgraphs by decreasing their frequencies below the predefined disclosure threshold. To do this, some edges are deleted from the chosen sensitive graphs. The aim of the EdgeDegree algorithm is to preserve the maximum number of non-sensitive patterns appearing in the original database (i.e., to reduce the information loss).

The EdgeDegree algorithm has also five steps as the EDH algorithm. The first four steps are the same with the EDH. But, local sanitization on a sensitive graph differs. That means, it also selects the sensitive graphs with less matches of the sensitive pattern set for sanitization, whereas it chooses the edges to be removed from a selected sensitive graph in a different way. Therefore, its pseudocode is similar to the Algorithm 1, but it calls Algorithm 3 at line 16.

The LocalSanitizationOfEdgeDegree algorithm takes a graph G and a sensitive pattern SP, and removes all the matches of the given pattern in the graph. For this, at first, the algorithm finds all the matches M. In a match of M, keys are graph vertex ids and values are sensitive pattern vertex ids. The keys and values in M are replaced, and M is

Algorithm 3 LocalSanitizationOfEdgeDegree(G, P).

```
1: M ← Determine the matches M(G, P)
2: Replace values with keys in each match of M
3: Sort M acc to values
4: while M is not empty do
5:     E ← list
6:     for each match m ∈ M do
7:         for (u, v) ∈ P.edges do
8:             (gu, gv) ← (m[u], m[v])
9:             Insert (gu, gv) into E
10:        end for
11:    end for
12:    mostFreqEdges ← Most frequent edges of G in E
13:    Create dictionary edgeRel
14:    for each edge ∈ mostFreqEdges do
15:        rel ← Sum of node degrees of edge
16:        Insert edge and rel pair into edgeRel
17:    end for
18:    minRelEdge ← Get the edge with min relevance from edgeRel
19:    Delete edge minRelEdge from G
20:    Remove M_e from M where M_e keeps the matches containing minRelEdge
21: end while
```

sorted based on values. The graph edges corresponding to the sensitive pattern edges in the matches are found. The most frequent edges $mostFreqEdges$ among these edges are identified. Then, the relevance of each edge in $mostFreqEdges$ is calculated by summing the degrees of its nodes. Next, the edge with minimum relevance $minRelEdge$ is deleted from the graph, and the matches with the edge are removed from the M. Local sanitization continues till there is no match in M.

Let us see the local sanitization on the graph given in Figure 4.1 for the pattern ($spid = 2$) whose matches are marked on the graph. The edges contained in the matches are the colored ones (green, red, or both colors). The most frequent edges are the edges with both colors since they are involved in the two matches and the others are included in only one match. From the most frequent ones, $(3, 8)$ and $(5, 9)$ have the minimum sum of node degrees, and $(3, 8)$ is selected as the victim edge. The $(3, 8)$ is deleted from the graph, and the matches containing it are removed. Since there is no match left, the sanitization completes.

4.1.3. Matchings & EdgeDegree Algorithm

The Matchings & EdgeDegree algorithm is an edge deletion-based subgraph hiding algorithm. The aim of this algorithm is to reduce the distance (i.e., to cause the minimum edge removal on the graph database) and to minimize the execution time. Thus, the algorithm employs the edgeMatches structure to identify the matches of more than one sensitive pattern containing an edge in one step. Since the identifiers of the matches are taken from the edgeMatches structure, the matches of the sensitive patterns are not found again while creating the sanitization information.

The Matchings & EdgeDegree algorithm consists of three main processes. The first process creates the structures from the database D. The second process creates the Sanitization Table (ST), which stores the modification information that will be applied to D. The last process is the sanitization process, in which a copy of the database is created and the sanitization information kept in the ST is applied to the copy of the database.

Creating Structures. The algorithm to create the structures for a given database D and a sensitive pattern set SP is illustrated in Algorithm 4. The edgeMatches structure is a 3-level dictionary, in which the keys are graph ids (gid), edges of the graph involved in the matches, and ids of sensitive patterns ($spid$) whose matches include the edge, respectively. For each sensitive pattern P in SP, each graph G in the database D is checked. All matches of the pattern in the graph (M) are found. The keys and values of a match in M are graph vertex ids and corresponding sensitive pattern vertex ids. The keys and values of the matches in M are replaced, and M is sorted based on values. Every match in M is inserted one by one as follows; for each graph $edge$ in the match, the match identifier is added into the edgeMatches to the active graph gid and pattern $spid$. If there was not entry for $edge$ or $spid$, it is created in edgeMatches before insertion. Besides, the grMatchCounts structure keeps the total match counts of each graph for the sensitive pattern set SP. It is going to be used to determine the order of the graphs to be sanitized. Additionally, the nodeAdj structure stores the number of neighbors of the nodes contained in the matches for each graph, not of all the nodes in the graph.

The edgeMatches structure for a graph ($gid = 254$ and represented in Figure 4.1) in Chemical dataset and the chosen sensitive pattern set from the dataset is denoted in Figure 4.3. The edges $(3, 8)$, $(3, 4)$, etc. are the graph edges included in the matches. The edge

Algorithm 4 Creating Structures.

Input: Database D, Sensitive patterns SP
Output: Edge Matches `edgeMatches`, Graph Match Counts `grMatchCounts`, Neighbors `nodeAdj`

Initialize dictionaries named `edgeMatches`, `grMatchCounts`, `nodeAdj`
for each $gid \in D.keys$ **do**
    Set `edgeMatches`$[gid]$ and `nodeAdj`$[gid]$ empty dictionaries
    `grMatchCounts`$[gid] \leftarrow 0$
end for
for each sensitive pattern $P \in SP$ **do** ▷ Set $P.matches$ and $G.matches$ to 0.
    $pMatchNum \leftarrow 0$
    for each $G \in D$ **do** ▷ gid and $spid$ are identifiers.
        $M \leftarrow$ Determine the matches $\boldsymbol{M(G,P)}$
        Replace values with keys in each match of M
        Sort M acc to values
        $lenMatches \leftarrow$ Number of matches in M
        for $i \leftarrow 1$ to $lenMatches$ **do**
            for $(u, v) \in P.edges$ **do**
                $edge \leftarrow (m[u], m[v])$ ▷ u is min, v is max.
                if $edge \notin$ `edgeMatches`$[gid].keys$ **then**
                    Set `edgeMatches`$[gid][edge]$ to an empty dictionary
                end if
                if $spid \notin$ `edgeMatches`$[gid][edge].keys$ **then**
                    Set `edgeMatches`$[gid][edge][spid]$ to an empty list
                end if
                Add i to `edgeMatches`$[gid][edge][spid]$
                if $u \notin$ `nodeAdj`$[gid]$ **then**
                    Set `nodeAdj`$[gid][u]$ to len of neighbors of u
                end if
                if $v \notin$ `nodeAdj`$[gid]$ **then**
                    Set `nodeAdj`$[gid][v]$ to len of neighbors of v
                end if
            end for
        end for
        $pMatchNum \leftarrow pMatchNum + lenMatches$
        `grMatchCounts`$[gid] \leftarrow$ `grMatchCounts`$[gid] + lenMatches$
    end for
    $P.matches \leftarrow pMatchNum$
end for

$(3, 8)$ is included in the sensitive patterns 2 and 4. The sensitive pattern 2 has two matches (match ids; 0 and 1) containing this graph edge.

Creating Sanitization Table. This process creates the Sanitization Table (ST) and updates the edgeMatches and nodeAdj structures. The algorithm for creating the ST is demonstrated in Algorithm 5. The ST keeps the list of edges to be removed from each

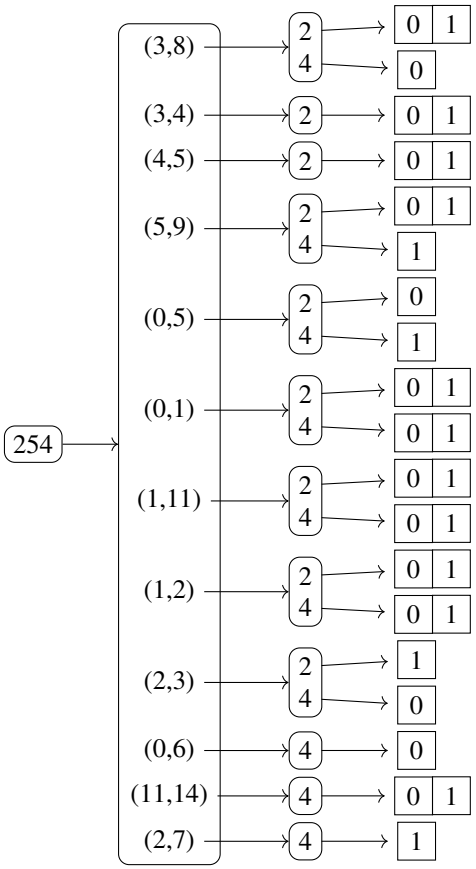


Figure 4.3. The representation of the edgeMatches structure used in the Matchings & EdgeDegree Algorithm.

graph. At the beginning of the process, the sensitive patterns and graphs are sorted in non-decreasing order of match counts. Among sensitive patterns in the set (SP), the pattern P with the least match count is selected first. The number of graphs to be sanitized is computed. Then, the P is checked for each graph with gid in such a way; For each edge involved in the matches of the P, the number of matches of P the edge is included is calculated from $edgeMatches$. The most frequent edges are determined according to P match counts. Then, for each candidate edge, count matches of all patterns in SP that the edge is contained using $edgeMatches$. This time the most frequent edges are identified according to SP match counts. Next, sum of node degrees of each candidate edge is calculated using $nodeAdj$, and the edge with the minimum total node degree is chosen as victim. After that, the ids of sensitive patterns and their match ids supporting the victim edge are uncovered from $edgeMatches$ and set to the variable $patternsMatches$. The $edgeMatches$ is updated by removing the match ids of sensitive patterns in $patternsMatches$ from other edges in $edgeMatches[gid]$, and deleting the victim edge from $edgeMatches[gid]$. The

Algorithm 5 Creating Sanitization Table.

Input: Edge Matches `edgeMatches`, Graph Match Counts `grMatchCounts`, Neighbors `nodeAdj`, Sensitive patterns SP, Disclosure threshold ψ
Output: Sanitization Table ST.

```
1: Initialize dictionary ST with an empty list for each gid
2: Sort gids in grMatchCounts in non-decreasing order of counts
3: Sort SP in non-decreasing order of P.matches
4: for each sensitive pattern P ∈ SP do
5:     NumGraphsForSanitization ← freq(P) − ψ + 1
6:     for i ← 1 to NumGraphsForSanitization do
7:         Get next gid such that graph with gid contains P
8:         while true do
9:             Set edgeMCounts to an empty dictionary
10:            for each edge ∈ edgeMatches[gid].keys do
11:                if spid ∈ edgeMatches[gid][edge].keys then
12:                    edgeMCounts[edge] ← len(edgeMatches[gid][edge][spid])
13:                end if
14:            end for
15:            if len(edgeMCounts) < 1 or max(edgeMCounts.values) < 1 then
16:                break
17:            end if
18:            mostFreqEdges ← Most frequent edges in edgeMCounts
19:            edgeTotalMCounts ← Get total match counts from
                   edgeMatches[gid] for each edge in mostFreqEdges
20:            mostFreqEdges ← Most frequent edges in edgeTotalMCounts
21:            Create dictionary edgeRel
22:            for each edge ∈ mostFreqEdges do
23:                rel ← Sum of node degrees of edge using nodeAdj
24:                Insert edge and rel pair into edgeRel
25:            end for
26:            victimEdge ← Get the edge with min relevance from edgeRel
27:            patternsMatches ← Determine sensitive pattern ids and their match
                   ids containing victimEdge using edgeMatches[gid]
28:            Remove match ids of sensitive patterns in patternsMatches from
                   other edges in edgeMatches[gid]
29:            Delete victimEdge from edgeMatches[gid]
30:            Decrease the values of nodes of victimEdge by 1 in nodeAdj
31:            Add victimEdge to ST[gid]
32:        end while
33:    end for
34: end for
```

values of the nodes of the victim edge are reduced by 1 in $nodeAdj$. The victim edge is added to the sanitization table ST for gid. Checking for sanitization on the graph gid for the pattern P completes when no edge remains in the graph or no matches remain.

To see how the local sanitization information is produced in Algorithm 5, let us

consider the edgeMatches in Figure 4.3. Suppose that the sensitive pattern P with $spid = 2$ is hidden on the given graph. The edges included in the P matches are $(3, 8)$, $(3, 4)$, $(4, 5)$, $(5, 9)$, $(0, 5)$, $(0, 1)$, $(1, 11)$, $(1, 2)$, and $(2, 3)$. For each edge, number of matches of P including the edge is counted. The most frequent edges according to P matches are $(3, 8)$, $(3, 4)$, $(4, 5)$, $(5, 9)$, $(0, 1)$, $(1, 11)$, and $(1, 2)$. Then, number of matches of all sensitive patterns including each candidate edge is counted. The most frequent edges according to SP matches are $(0, 1)$, $(1, 11)$, and $(1, 2)$. Now, their node degree sum values are counted. For these edges, all values are equal, and $(0, 1)$ is selected as the victim edge. The sensitive pattern ids and their match ids including the $(0, 1)$ are uncovered from $edgeMatches$ (the match ids 0 and 1 of the pattern 2 and the match ids 0 and 1 of the pattern 4) and put into the $patternsMatches$ variable. The match ids of each pattern in $patternsMatches$ are deleted from the match id list of other edges. For this example, deleting the match ids eliminates all the matches.

Sanitizing the Database. The final process of the Matchings & EdgeDegree algorithm is sanitizing the database. This process creates a copy of the database D, and uses the sanitization table (ST) produced in the previous process to apply the modification information to this copy and get the sanitized database D'.

4.1.4. Subgraph Hiding with Edge Entropy

Sen et al. (*88*) formalize the edge entropy to rank the edges in a graph and find the important edges. In that study, the edges have meaningful information and are weighted. However, the edges in the graph may not have the weights like in functional brain graphs or uncertain graphs. Moreover, Qiao et al. (*87*) quantify the influence of nodes using an entropy model. The influence of a node is found by calculating the local influence of it on one-hop neighbors and the indirect influence of it on two-hop neighbors. They deal with the entropy for only the nodes.

In this thesis, the edge entropy of an edge in a given graph is calculated as follows; First, a subgraph consisting of one-hop subgraphs of the incident nodes of the edge is constructed. Because the edges are not associated with the weights, the degree-based graph entropy as in Equation 2.19 is used to define the edge entropy. But, the size of the subgraph is different because the subgraph is formed according to the edge. Hence, the edge entropy

of an edge is defined as

$$H_e = \log\left(\sum_{i=1}^{n_S} SDC_i\right) - \sum_{i=1}^{n_S} \frac{SDC_i}{\sum_{j=1}^{n_S} SDC_j} \log SDC_i, \tag{4.1}$$

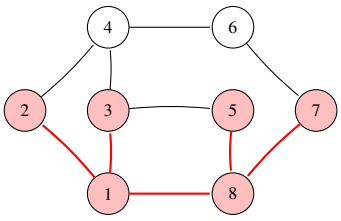
where n_S is the size of the subgraph and SDC_i is the degree of node i in the subgraph.

In Figure 4.4, an example for the calculation of the edge entropy of the edge $(1, 8)$ on the graph is displayed. The subgraph colored in the red is constructed. The SDC_i values of the nodes in the subgraph can be seen from the figure. The edge entropy is calculated as

$$H_{(1,8)} = \log\left(\sum_{i=1}^{6} SDC_i\right) - \sum_{i=1}^{6} \frac{SDC_i}{\sum_{j=1}^{6} SDC_j}.$$

Two algorithms are proposed for the subgraph hiding problem using the edge entropy introduced in Equation 4.1. These algorithms are EdgeEntropy and Matchings & EdgeEntropy. The EdgeEntropy algorithm works similarly to the EdgeDegree algoritm, but it differs in the local sanitization. Rather than calculating the edge degree at the line 15 in LocalSanitizationOfEdgeDegree (Algorithm 3), it calculates the edge entropy.

As the Matchings & EdgeDegree algorithm, the Matchings & EdgeEntropy algorithm is composed of three processes, which are creating the structures, creating the sanitization table, and sanitizing the database. However, there are some differences in the first two processes. In the process of creating the structures, the nodeAdj structure keeps the neighbors of the nodes contained in the matches for each graph, instead of the number of neighbors. The $edgeMatches$ and $grMatchCounts$ are created as in the Matchings & EdgeDegree algorithm. With $edgeMatches$ structure, this algorithm also does not need to find the matches of a sensitive pattern in a graph while producing the sanitization information. That means, graph matching is only performed during the creation of the



Node	1	2	3	5	7	8
SDC_i	3	1	1	1	1	3

Figure 4.4. An example showing the calculation of the edge entropy of the edge $(1, 8)$ on the graph, and the SDC_i values of the nodes in the formed subgraph.

structures. In the process of creating the sanitization table, while checking a sensitive pattern P on a graph to get the sanitization information, the followings are done; edges in matches of the P are chosen firstly according to the number of inclusions in the pattern P matches, and then according to the number of inclusions in the sensitive pattern set SP matches. After the most frequent edges are determined, rather than finding the edge degree, the edge entropy of each candidate edge is calculated using $nodeAdj$, and the edge with the minimum edge entropy is chosen as the victim. The remaining steps are the same except that nodes of the victim edge are deleted from the list of each other in $nodeAdj$.

4.2. Community Hiding Methods

A variety of methods have been devised to identify overlapping communities in networks, enabling nodes to belong to multiple communities and better representing the real-world systems. Among these, there are many overlapping community detection algorithms designed using the concept of cliques, which can serve as the fundamental units of communities. Recognizing this, we propose strategically manipulating the links in the network to diminish the effectiveness of overlapping community detection algorithms, particularly those that rely on clique structures or local expansion methods. Disrupting cliques weakens the core elements of such algorithms, impairing their ability to accurately identify overlapping communities.

4.2.1. 4-clique Community Hiding (4clqCH)

A clique is a set of nodes that are all linked to each other through edges, forming a complete subgraph. A k-clique is a clique of size k, that is, every pair of k vertices is linked by an edge. The study (*130*) points out that several community detection methods based on k-cliques (*5*, *53–55*, *131–134*) have been developed. In this thesis, 4-clique Community Hiding, called 4clqCH, is proposed as a global community hiding algorithm. Its aim is to disrupt the most common edges in the 4-cliques, thus changing the community structure relying on the cliques as much as possible.

The 4clqCH algorithm first counts the number of 4-cliques that each edge is a part of. For this, the fast method proposed by the study (*135*) is used. The counting method

employs degree ordering for clique counting. The degree ordering of the graph G is indicated by $\prec$. For vertices i and j, $i \prec j$ means that $deg(i) < deg(j)$ or $deg(i) = deg(j)$ and $i < j$ (comparing their vertex ids). The $degree\ ordered$ DAG (directed acyclic graph) is constructed by orienting all edges of G based on $\prec$.

The pseudocode of the 4clqCH is depicted in Algorithm 6. Line 1 in the pseudocode creates the degree ordered DAG, namely $Gout$, from the input graph. Every 4-clique in the original graph is counted once. The $per\text{-}edge$ counts for 4-clique pattern are calculated as follows (Lines 3-18); All edges (i, j) are iterated, where $i \prec j$. All the out-wedges containing (i, j) are enumerated, and all triangles containing (i, j) with i as the smallest vertex are found. All pairs of such triangles formed by (i, j) are iterated. For each pair, the missing edge is checked whether it forms a four-clique. Figure 4.5 shows core patterns for 4-clique counting.

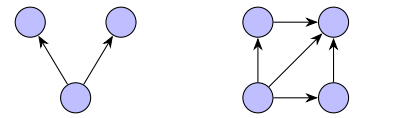


Figure 4.5. Core patterns for 4-clique counting (Source: (*135*)).

After counting, the edges with the highest count is detected from $per\text{-}edge$ counts (Lines 20-21). The edge that has the min degree among the candidate edges is selected as the victim edge (Line 22). Next, the 4-cliques that include the victim edge are determined. The $per\text{-}edge$ count value of edges involved in these matches is reduced by 1 (Line27). The victim edge is deleted from the graphs (Line 31).

Algorithm 6 4clqCH Algorithm.

Input: Graph G, Budget β
Output: Modified graph G'.

```
1: Create degree ordered Gout from G
2: Create arrays triends of size |V| + 1, perEdge of size |E| + 1
3: for each vertex i ∈ V do
4:     for each out-neighbor j of i do
5:         for each next out-neighbor k of i do
6:             if edge (j, k) exists then
7:                 Insert k into triends
8:             end if
9:         end for
10:        for each vertex k in triends do
11:            for each next vertex l in triends do
12:                if edge (k, l) exists then
13:                    Increase perEdge of edges formed with {i, j, k, l} by 1
14:                end if
15:            end for
16:        end for
17:    end for
18: end for
19: for t = 0 to β do
20:    Find maxFreq in perEdge                          ▷ if 0, use triangles
21:    mostFreqs ← Get edges with maxFreq
22:    (i, j) ← Find edge with min degree in mostFreqs                ▷ (i, j) is victim
23:    commonNbrs ← Find common neighbors of i and j (in G)
24:    for each k in commonNbrs do
25:        for each next l in commonNbrs do
26:            if edge (k, l) exists then
27:                Decrease perEdge of edges formed with {i, j, k, l} by 1
28:            end if
29:        end for
30:    end for
31:    Update G and Gout by deleting edge (i, j)
32: end for
33: G' ← G
```

CHAPTER 5

PERFORMANCE EVALUATION OF THE PROPOSED METHODS

This chapter is dedicated to the performance evaluation of the graph hiding methods introduced in this thesis, providing a critical assessment of their effectiveness in achieving the desired task. The first section presents the evaluation of the algorithms developed for subgraph hiding. The second section presents the evaluation of the global community hiding algorithms.

5.1. Performance Evaluation of Subgraph Hiding Algorithms

In this section, performance evaluation of subgraph hiding algorithms is given. First of all, the EDH algorithm (*15*) is compared with the three vertex masking algorithms (*9*) Heuristic1, Heuristic2 and Heuristic3, named VMask1, VMask2 and VMask3, respectively. Then, the edge deletion-based subgraph hiding algorithms proposed in this thesis (EDH, EdgeDegree, Matchings & EdgeDegree, EdgeEntropy and Matchings & EdgeEntropy) are compared. All the algorithms including the masking ones are implemented in Python. The experiments are carried out on a laptop with an Intel Core i7 8750 2.2 GHz CPU, 16GB RAM, and Windows 10 operating system.

Table 5.1. The properties of graph datasets used to evaluate subgraph hiding algorithms.

	Chemical	Movielens	NCI109
$\|D\|$	340	802	4127
$\|V\|$	9189	29001	122494
$\|E\|$	9317	202263	132603
Avg. vertices	27	36.2	29.7
Avg. edges	27.4	252.2	32.1
# of vertex labels	66	5800	38
# of edge labels	4	1	3

5.1.1. Datasets

The experiments are conducted on three datasets, which are Chemical (*136*), Movielens (*137*) and NCI109 (*138*). Their properties are specified in Table 5.1. In the Chemical dataset, graphs represent chemical compounds where vertices indicate atoms and edges indicate bonds between atoms. Vertex labels correspond to the atom types and edge labels correspond to the bond types. The Movielens dataset contains movie tags given by users along with corresponding assignment times. It is a single graph that has 47,957 tag assignments within a specific time frame. A graph is generated for each day based on the single graph. In these daily graphs, each tagged movie represents a vertex, and an edge is added between two movies if they share the same tag. Each vertex is labeled with a movie id, while the edges are unlabeled. Hence, '0' is used as the label and the number of distict edge labels is written as 1 in the table. After removing the graphs without edges, the dataset has a total of 802 graphs. The NCI109 dataset is a widely used cancer screening dataset.

Frequent patterns are extracted from the datasets using gspan algorithm (*29*). For each dataset, five of these patterns are selected randomly as sensitive patterns. Figure 5.1 illustrates the chosen sensitive patterns from the datasets Chemical, Movielens, and NCI109 that are located at the top, middle, and bottom, in order. Each sensitive pattern is accompanied by its frequency. The VF2 algorithm (*139*) is used to find the matchings of a subgraph in a graph. Experiments are conducted to assess how different disclosure thresholds impact algorithm performance on various datasets. Each result is the average of five runs. The support and disclosure thresholds are equal when assessing information loss and artifact patterns. Movielens experiments are conducted with disclosure thresholds ranging from 7 to 12 because frequent patterns cannot be identified using a threshold of 6 on our computer, and the least frequent sensitive pattern selected from it occurs 12 times.

5.1.2. Experimental Results of the EDH and Masking Algorithms

The study (*9*) measures the runtime, number of masking symbols, and percentage of preserved frequent patterns. In the EDH and masking algorithms, hiding failure is zero. The algorithms are evaluated in terms of execution time, information loss, artifact patterns, and distance.

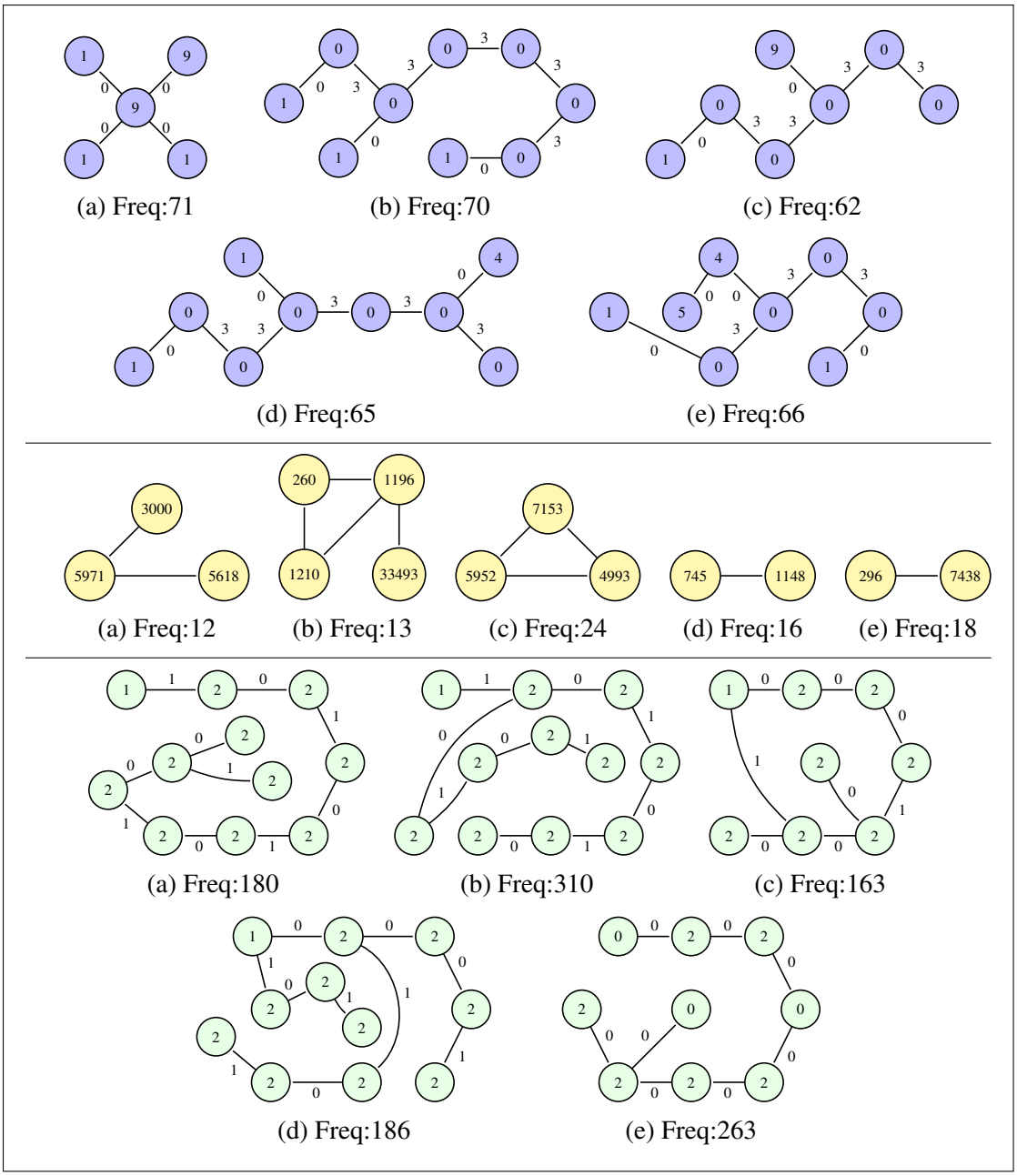


Figure 5.1. Selected sensitive patterns from the datasets Chemical, Movielens and NCI109, respectively.

Figure 5.2 displays the execution times (ms) of VMask1, VMask2, VMask3, and EDH algorithms on the datasets Chemical, Movielens, and NCI109 across different threshold values. VMask1 performs the best due to not considering the matchings of a sensitive pattern in a sensitive graph, avoiding the need for a subgraph matching algorithm. On the other hand, VMask3 shows the longest execution times as it may require multiple runs of a subgraph matching algorithm. VMask2 and EDH demonstrate comparable execution times. It is observed that execution times decrease with higher disclosure

thresholds due to fewer graphs being sanitized. The differences in results are more for the Chemical and NCI109 datasets compared to the Movielens dataset, where each movie id (vertex label) occurs at most once in a graph, causing at most one match for a sensitive pattern. Additionally, the patterns in the Movielens dataset are with low support values.

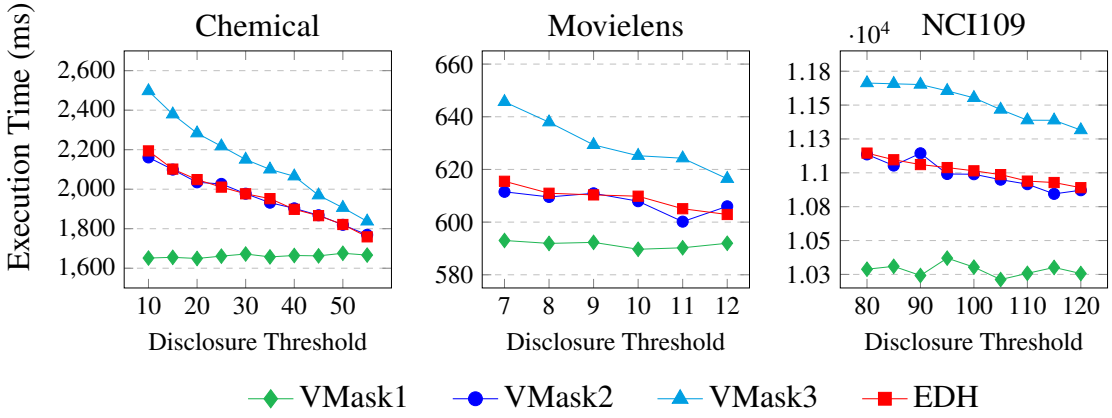


Figure 5.2. Execution time (ms) results of the EDH and masking algorithms for different disclosure threshold values.

The information loss evaluates the proportion of unintentionally hidden non-sensitive frequent patterns during sanitization. Figure 5.3 shows the information loss (%) of the algorithms on the datasets Chemical, Movielens, and NCI109 at various threshold values. The EDH algorithm outperforms the others across all datasets by deleting edges rather than masking vertex labels. Masking a vertex label means that the associated edges are also affected. This is because non-sensitive patterns present in the original data involving that vertex are no longer extracted, indirectly impacting its edges. But, when EDH deletes an edge, most non-sensitive patterns can still be extracted using the other edges. Thus, it shows the best performance in preserving of the non-sensitive patterns. The ratio changes from approximately 1.5 to 2.5 for VMask2 and VMask3, and from 2 to 4 for VMask1 in the Chemical dataset. In the Movielens dataset, the ratio is around 2 for disclosure thresholds between 7 and 9, and decreasing as the threshold approaches 12. In the NCI109 dataset, the ratio is about 3 for VMask2 and VMask3, and increasing to almost 4 for VMask1.

Masking the vertex labels in the sensitive graphs with a symbol that is not part of the labels for vertices and edges leads to lots of vertices with this symbol, and frequent patterns with this symbol emerge after sanitization. Figure 5.4 shows the artifact patterns

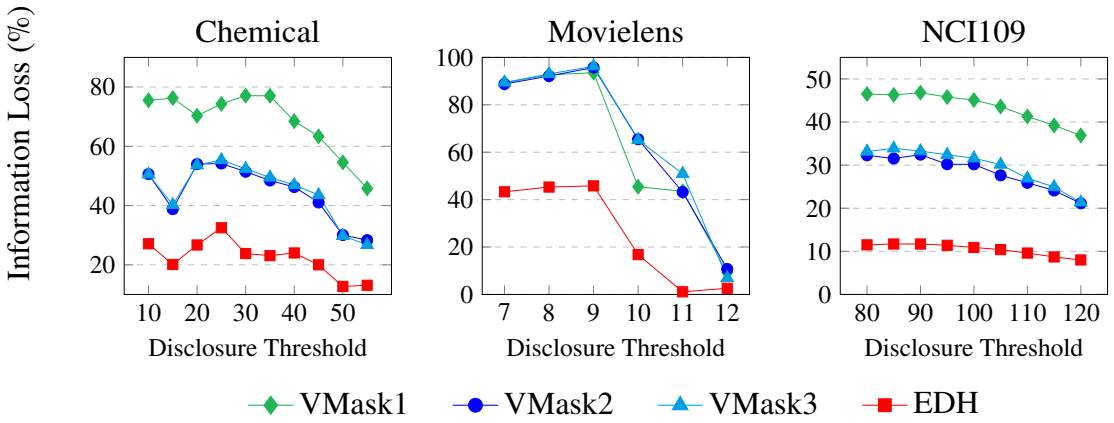


Figure 5.3. Information loss (%) results of the EDH and masking algorithms for different disclosure threshold values.

produced by the algorithms for distinct disclosure thresholds on the datasets. The EDH method does not create artifact patterns, unlike the masking algorithms because EDH conceals sensitive patterns by deleting the selected edges so it does not employ a masking symbol. Consequently, no new patterns emerge after the hiding process. The number of artifact patterns generated by the other algorithms decreases as the disclosure threshold ascends since fewer masking symbols are added when sanitizing fewer graphs.

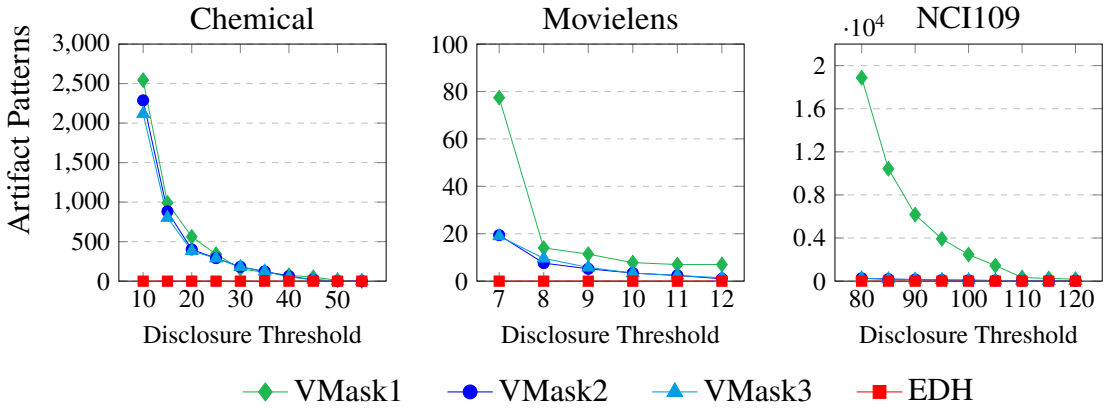


Figure 5.4. Artifact patterns results of the EDH and masking algorithms for different disclosure threshold values.

The original and sanitized databases should be similar with the minimal distance between them. The distance is determined by counting the altered vertices and edges. In the masking methods, the distance is the total masked vertices, and in EDH, it is the total deleted edges. Figure 5.5 shows the distance results of the algorithms for the datasets,

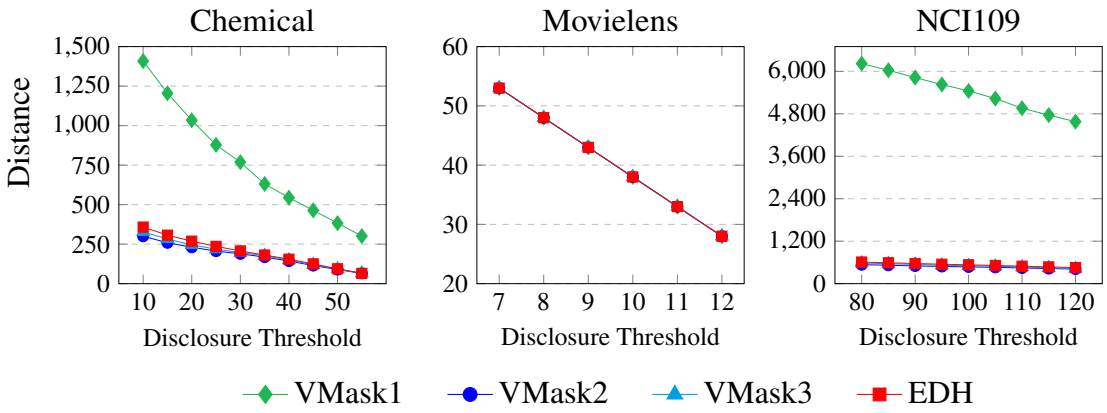


Figure 5.5. Distance results of the EDH and masking algorithms for different disclosure threshold values.

which are lower at higher disclosure thresholds due to sanitizing less graphs. In spite of the close results, touching (masking/deleting) an edge has a smaller impact on the graph structure compared to a vertex. Further, the values are the same across all algorithms for the Movielens dataset.

5.1.3. Experimental Results of the Edge Deletion-Based Algorithms

Because all the proposed edge deletion-based subgraph hiding algorithms (EDH, EdgeDegree, Matchings & EdgeDegree, EdgeEntropy and Matchings & EdgeEntropy) do not cause the hiding failure and do not produce fake patterns, they are evaluated in terms of execution time, information loss and distance.

Figure 5.6 demonstrates the execution times (ms) of the five algorithms on the datasets Chemical, Movielens and NCI109 for distinct disclosure thresholds. The EDH, EdgeDegree and EdgeEntropy algorithms exhibit the similar results because the latter ones only change the local sanitization to choose the victim edge. On the Chemical and NCI109 datasets, the Matchings & EdgeDegree and Matchings & EdgeEntropy algorithms outperform the other algorithms since they employ the edgeMatches structure during the process of creating the sanitization information, and find the matches only one time. On the Movielens dataset, the execution times of the Matchings-based algorithms are worse than the others because the average number of vertices and the average number of edges in this dataset are more than those of the other datasets, that can be seen in Table 5.1.

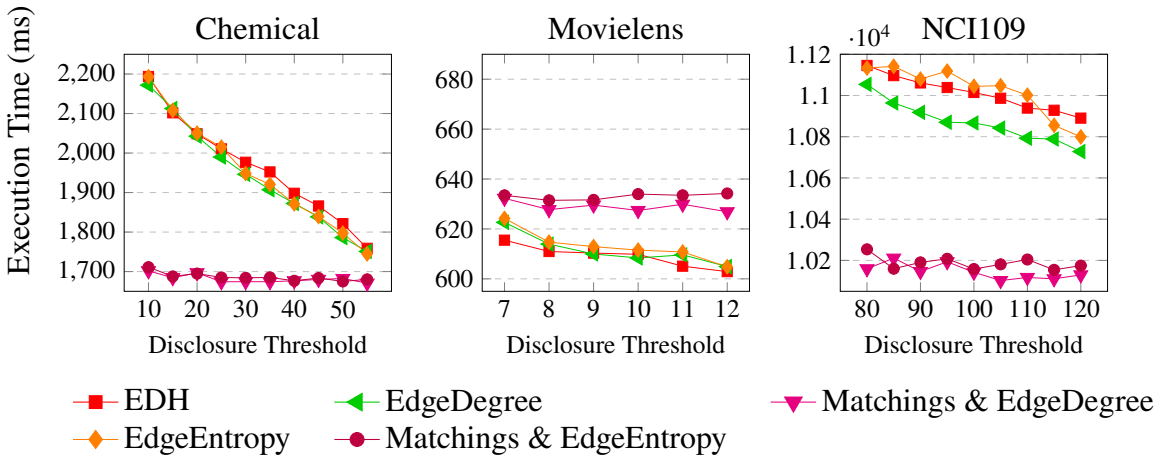


Figure 5.6. Execution time (ms) results of the edge deletion-based algorithms for different disclosure threshold values.

Figure 5.7 shows the information loss (%) of the five algorithms on the three datasets for varying disclosure thresholds. On the Chemical dataset, the EdgeDegree and EdgeEntropy algorithms produce the same results and are slightly better than the EDH algorithm. Then, the Matchings & EdgeDegree and Matchings & EdgeEntropy algorithms generate the same results for the information loss on this dataset. On the Movielens dataset, the EDH yields the different results from all the other algorithms and causes less losses than the others. However, the Movielens dataset represents movies and tags, and each graph includes a movie id (label) only once. Thus, there is one matching in a graph for a sensitive pattern. Additionally, the supports of patterns are very low in this dataset. Therefore,

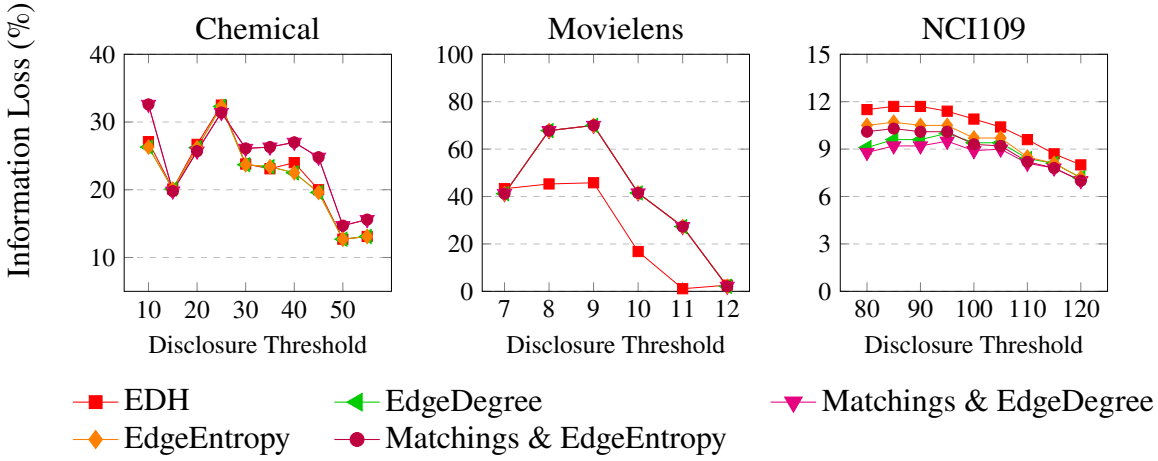


Figure 5.7. Information loss (%) results of the edge deletion-based algorithms for different disclosure threshold values.

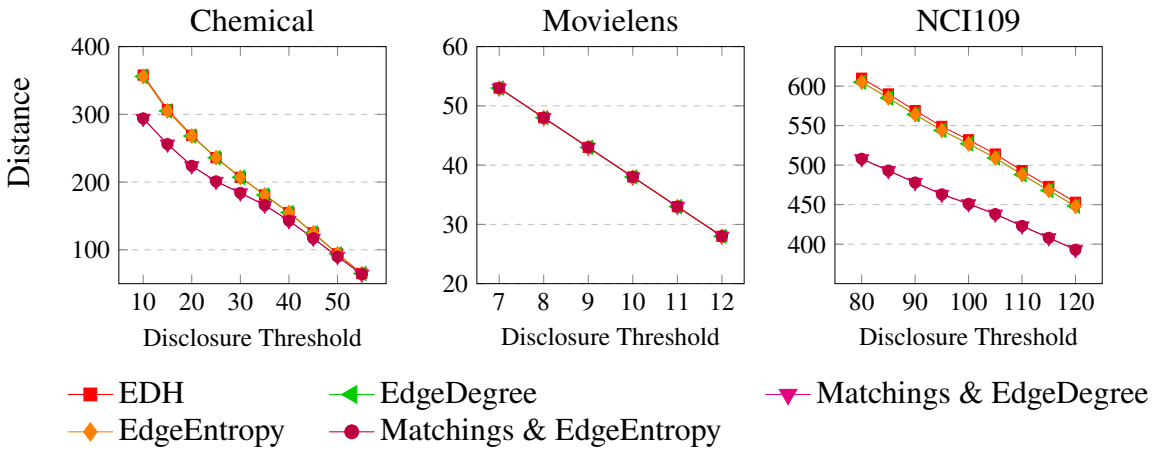


Figure 5.8. Distance results of the edge deletion-based algorithms for different disclosure threshold values.

removing an edge critically affects the results. On the NCI109 dataset, the Matchings & EdgeDegree algorithm outperforms the other algorithms for all the disclosure thresholds. Other subgraph hiding algorithms proposed in this study are better than the EDH according to the information loss on this dataset.

Figure 5.8 illustrates the distance between the original and sanitized databases on the three datasets. On the Chemical dataset, the Matchings-based algorithms yield the better results than the other algorithms. On the Movielens dataset, all the algorithms result in the same distance. On the NCI109 dataset, the Matchings & EdgeDegree and Matchings & EdgeEntropy algorithms have the same results, and they have better results than the other three algorithms. Further, the EdgeDegree and EdgeEntropy algorithms give the same results and are better than the EDH for this dataset. Reducing (deleting) less edges means the smaller impact on the database. Because the idea of the Matchings-based algorithms is based on that if sensitive patterns share a common edge, removing this edge eliminates more than one sensitive pattern at the same time.

5.2. Performance Evaluation of Community Hiding Algorithms

We conduct a comparative analysis to systematically evaluate the results of the 4clqCH algorithm against those produced by Random-Rewire (*8*), LSHA (*126*), importance measure-based methods (Degree, Betweenness, and Closeness) (*122*), and Random-Del. The motivation for choosing these community hiding algorithms is that (i) they can work

independently of the result produced by a community detection method during the hiding process, and (ii) they are heuristic approaches.

In our experiments, we employ the Python implementation of the LSHA algorithm provided by its authors and correct it by appropriately setting the local structure index of the deleted edge. As the clique counting method (*135*), the 4clqCH algorithm is also implemented in C++. Since the code of the other algorithms is not publicly available, we also implement all these algorithms (Random-Rewire, Degree, Betweenness, Closeness, and Random-Del) in C++. The details of the centrality definitions are not given in the study (*122*), thus edge degree is considered as the sum of node degrees, and edge closeness is calculated according to the definition in the study (*140*). The interactive version of these algorithms is considered, that is, the updated value of the centrality is taken into account after an edge update.

The 4clqCH is compared with the chosen algorithms (Random-Rewire (*8*), LSHA (*126*), Degree (*122*), Betweenness (*122*), Closeness (*122*), and Random-Del) regarding normalized mutual information (NMI) and overlapping NMI (ONMI) depending on the disjoint and overlapping nature of the community detection algorithm (refer to Figures 5.10 – 5.14). In addition to the overlapping community detection algorithms, such as Clique Percolation Method (CPM) (*54*), PercoMCV (*55*), UMSTMO (*56*), Demon (*57*), GREESE (*58*), the disjoint algorithm Core Expansion (*141*) is chosen due to its reliance on a seed-based approach. The UMSTMO, Demon and GREESE are chosen to fool although they are not clique-based methods. To find the k-clique communities with CPM, the parallel implementation (COS) (*142*) is used to solve the scalability issues.

5.2.1. Community Detection Algorithms

The functionalities of the community detection algorithms are explained to provide a more comprehensive understanding of how well the community hiding methods succeed in obfuscating the community structure.

CPM (*54*) is a clique-based algorithm to detect overlapping communities in networks. It characterizes a k-clique community as the union of all k-cliques that are connected through a sequence of adjacent k-cliques, where adjacency is defined by the sharing of k-1 nodes. The process of identifying k-clique communities begins with locating all cliques in

the network. Subsequently, analysis is performed on the clique-clique overlap matrix to determine the community structure.

PercoMCV (*55*) is a hybrid method. It begins by leveraging the CPM (*54*) to identify initial community structure. Following this, eigenvector centrality is applied to the preliminary results to minimize the proportion of unclassified nodes.

Core Expansion (*141*) first detects the core of each potential community in the network. The core is then expanded iteratively by adding nodes to generate the final communities. It is based on the neighborhood overlap, which is formalized for an edge (u, v) as $c_{uv}/(d_u + d_v - c_{uv} - 2)$, where d_u and d_v are the degrees of nodes u and v, respectively, and c_{uv} is the number of common neighbors of u and v. The initial cores are local maximum nodes, where the weight of a node is the sum of the neighborhood overlap values of its edges.

UMSTMO (*56*) leverages the union of maximum spanning trees (UMST) as its foundational approach. The method operates in three distinct phases: first, the construction of the UMST T; second, the identification of local communities from T, wherein nodes A, B, and C are considered part of the same community if nodes B and C are connected to A within the UMST T and collectively form a triangle in the original graph G; and third, the integration of preliminary communities based on the importance of their shared nodes.

Demon (*57*) identifies overlapping communities by analyzing the ego networks (local neighborhoods) of nodes. The local communities for a node are found with the Label Propagation Algorithm (*53*), which are then merged according to the similarity threshold to obtain the global communities.

GREESE (*58*) employs a coupled-seed expansion approach for detecting overlapping communities. A coupled-seed is formed by selecting a node along with its most similar neighboring node, serving as the starting point for expansion. This seed is then iteratively expanded using a fitness function designed to enhance the detection of local communities. Ultimately, local communities with substantial overlap are merged to produce the final set of communities.

5.2.2. Datasets

Different real world datasets are used from the Network Repository (*143*) and SNAP (*144*) platforms, each exhibiting distinct characteristics. Table 5.2 summarizes their properties, such as the number of nodes, the number of edges, the average clustering coefficient, and the size of the maximum clique. We make a graph undirected if it is directed, and the self-loops and multiple edges are removed.

Table 5.2. The properties of the datasets used to evaluate community hiding algorithms.

Graph	Nodes	Edges	Avg. CC	Max clique
Karate	34	78	0.571	5
Polbooks	105	441	0.487	6
Football	115	613	0.403	9
Email	1,133	5,451	0.254	12
CA-GrQc	5,242	14,496	0.530	44
Amazon	262,111	1,234,877	0.420	7
Youtube	1,134,890	2,987,624	0.081	17

Figure 5.9 illustrates the frequency distribution of k-cliques within the datasets. Specifically, the x-axis represents the size of the cliques (k), while the y-axis indicates the corresponding count of k-cliques present in the dataset. This visualization provides an overview of the prevalence of cliques of various sizes in the datasets.

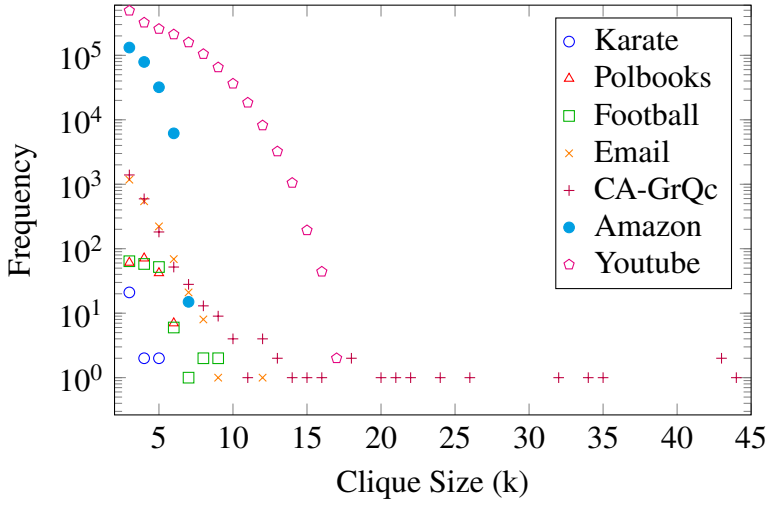


Figure 5.9. Clique distribution in each graph.

5.2.3. Experimental Results of the 4clqCH Algorithm

This section presents the discussion regarding the results of the 4clqCH algorithm in comparison to the other algorithms, as well as the complexities of the algorithms.

Comparison of the algorithms: The 4clqCH algorithm produces low ONMI scores across most scenarios when evaluated against the CPM and PercoMCV for the datasets karate, polbooks, football, and email. This is primarily due to its targeted approach of disrupting cliques, particularly 4-cliques, which significantly diminishes the ability of clique-based algorithms, such as CPM and PercoMCV. Additionally, it demonstrates strong performance in deceiving the Core Expansion method across these datasets. Since the Core Expansion relies on neighborhood overlap, the 4clqCH disrupts critical edges in localized regions, thereby reducing neighborhood overlap scores and making it harder for the Core Expansion to identify and expand the cores. Similarly, the Degree (*122*) algorithm, which focuses on removing high-degree edges, shows some good effectiveness in deceiving the clique-based methods and Core Expansion (on karate, polbooks, and email), meaning that high-degree edges are often included within cliques and are important in terms of neighborhood overlap.

The GREESE algorithm operates based on common neighbors and functions similarly to weak clique percolation. The 4clqCH algorithm is highly effective in most scenarios for reducing the ONMI scores of the GREESE algorithm. This effectiveness stems from its strategy of removing edges that are frequently shared among 4-cliques, which play a crucial role in ensuring high neighborhood overlap and supporting the expansion of communities.

When the effect of the community hiding algorithms to the Demon algorithm is analyzed on these datasets, it is observed that none of them is superior to the other. Moreover, the hiding algorithms effectively disrupt the functionality of the Demon algorithm on email dataset, indicating that the targeted alterations to edges appear to significantly perturb the structures of the ego neighborhoods and overlapping substructures that the Demon relies upon for its community detection process.

There is no consistency in the success of fooling the UMSTMO algorithm (except for the Betweenness (*122*) algorithm), that is, the effectiveness of the community hiding algorithms varies according to the characteristics of the datasets. While the 4clqCH

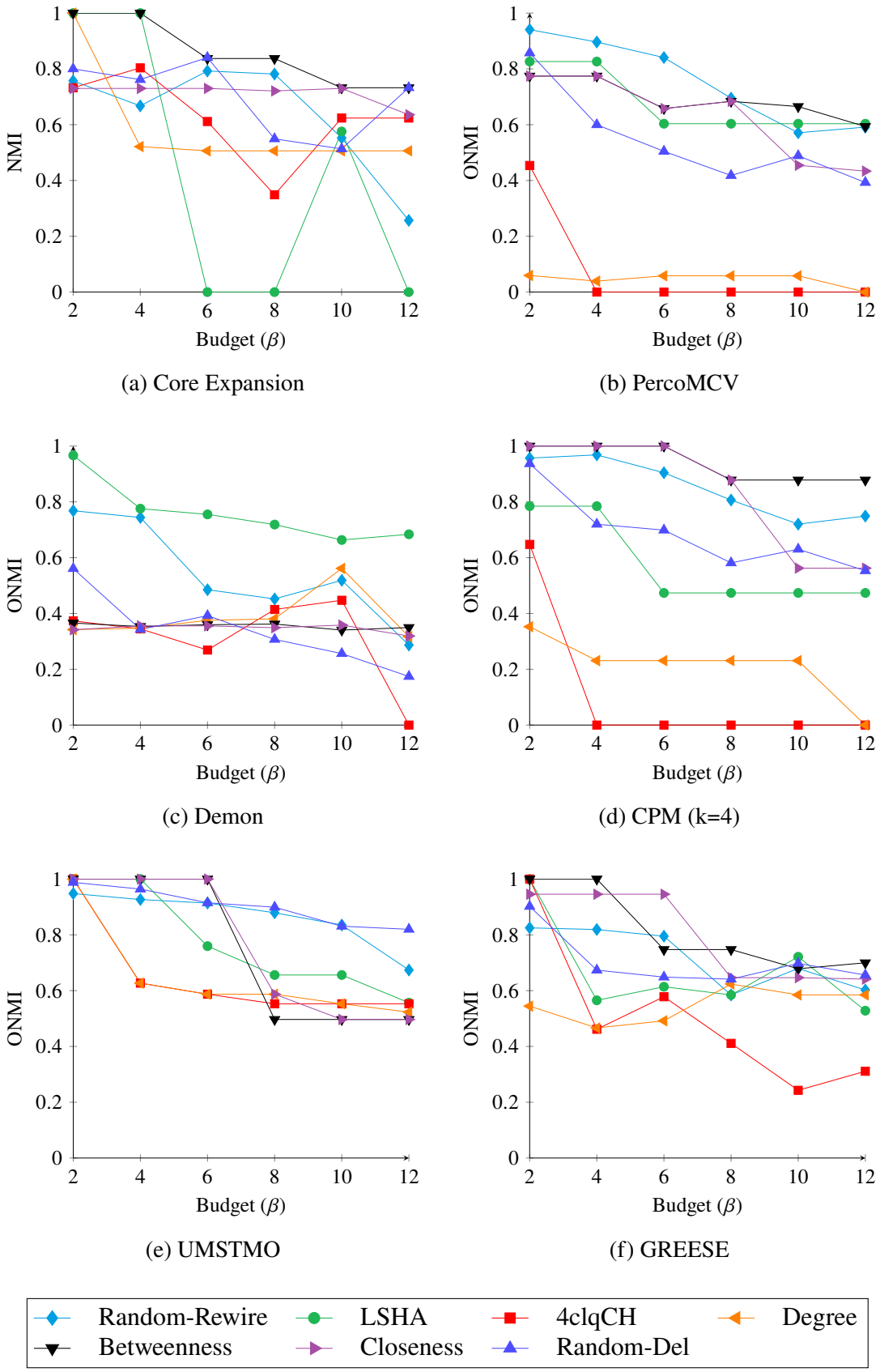


Figure 5.10. NMI/ONMI of community hiding algorithms according to the budget on karate dataset.

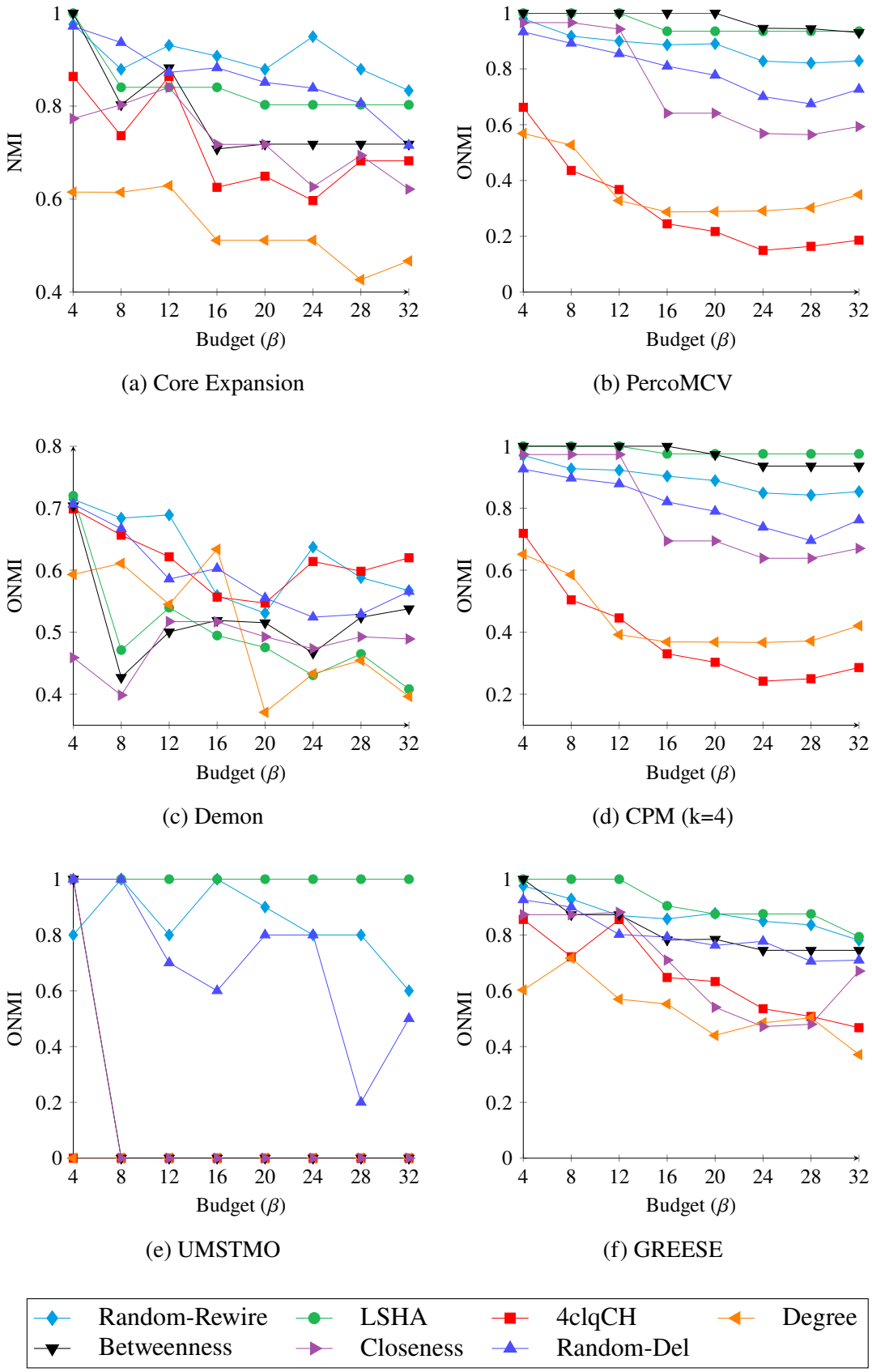


Figure 5.11. NMI/ONMI of community hiding algorithms according to the budget on polbooks dataset.

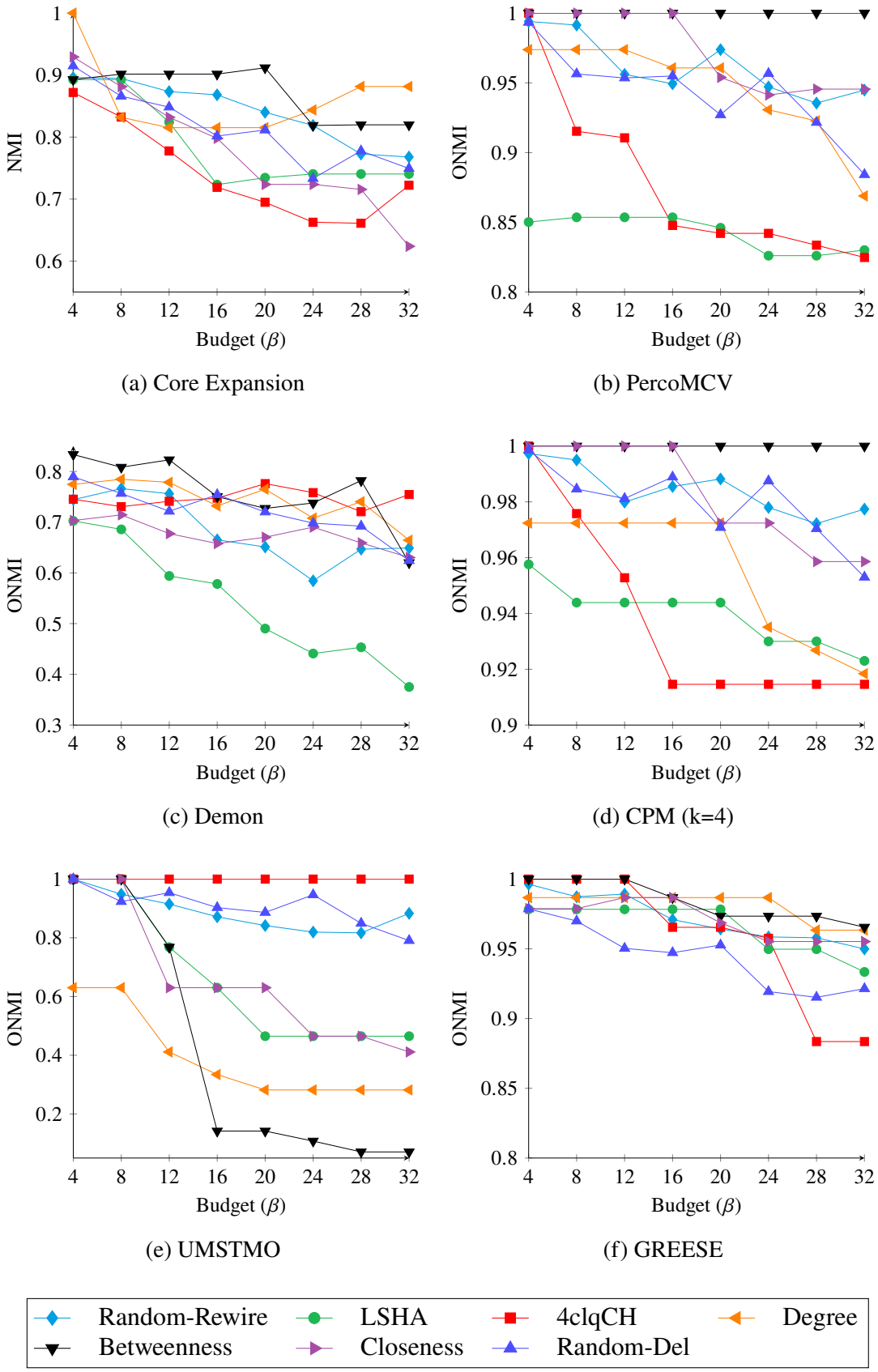


Figure 5.12. NMI/ONMI of community hiding algorithms according to the budget on football dataset.

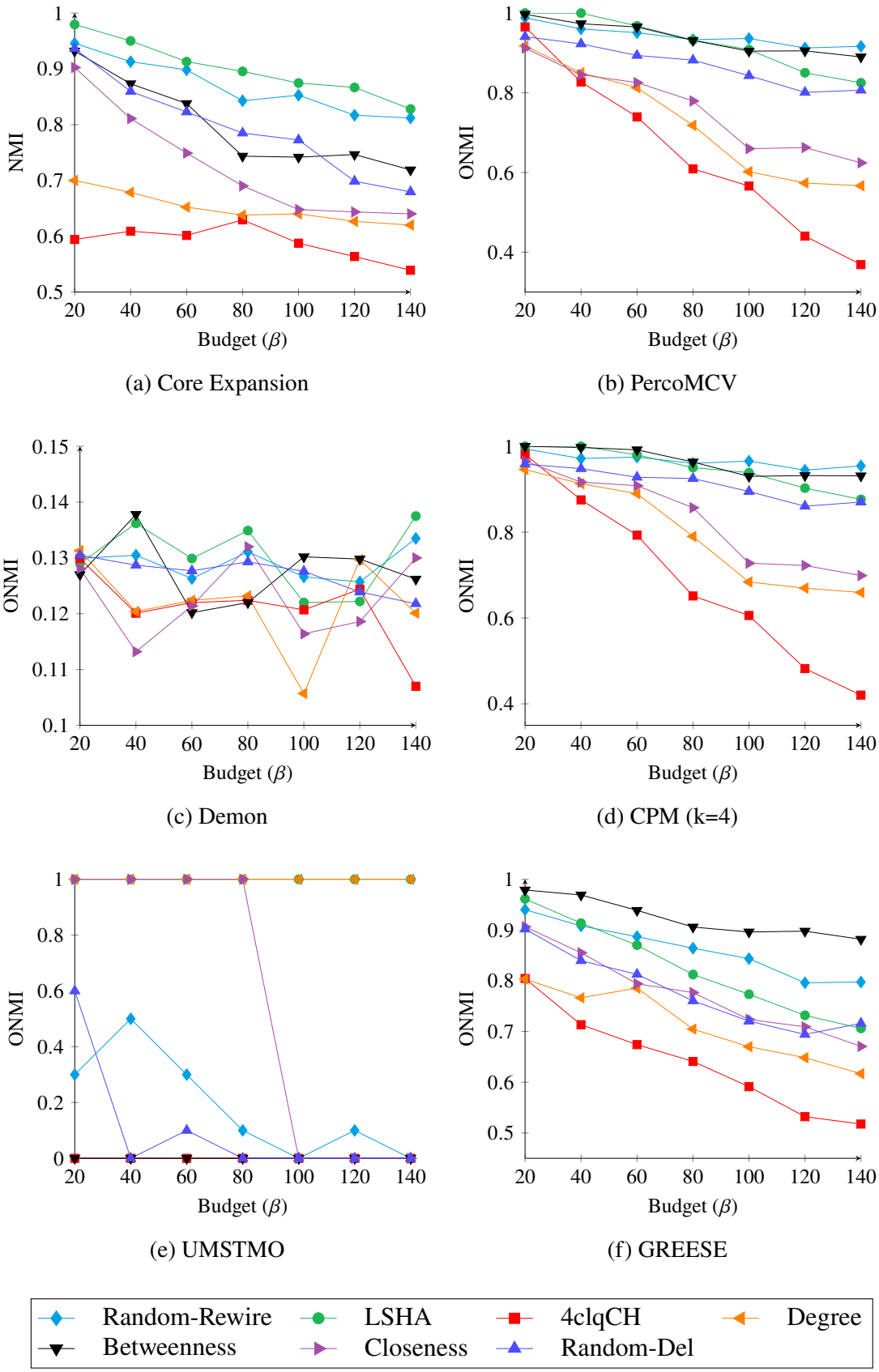
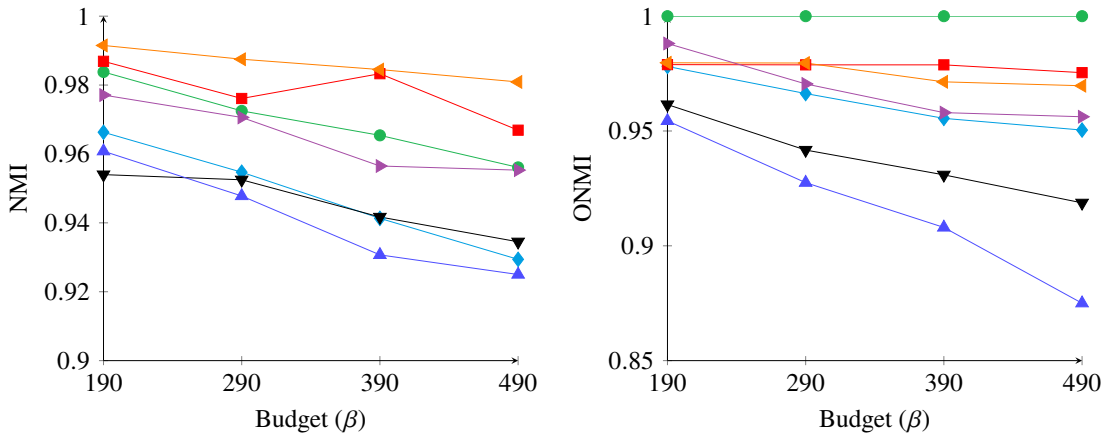
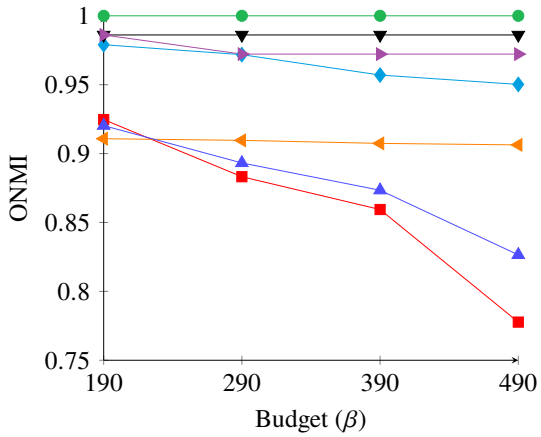


Figure 5.13. NMI/ONMI of community hiding algorithms according to the budget on email dataset.

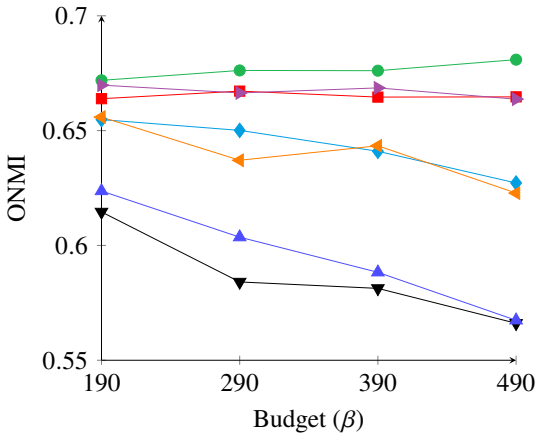


(a) Core Expansion

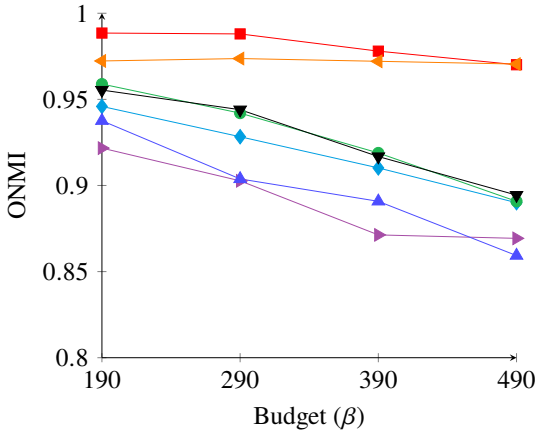
(b) CPM (k=4)



(c) CPM (k=7)



(d) Demon



(e) GREESE

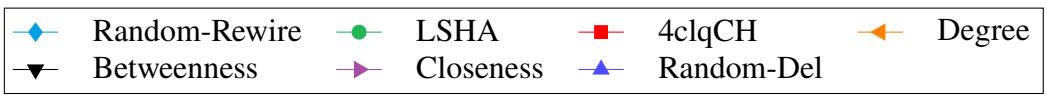


Figure 5.14. NMI/ONMI of community hiding algorithms according to the budget on CA-GrQc dataset.

algorithm performs effectively on the karate, polbooks, and email datasets, it fails to deceive the UMSTMO on the football dataset. The Betweenness algorithm, which removes edges with the highest betweenness centrality, focuses on edges bridging different communities and maintaining network connectivity. Removing these edges can fragment the network, directly affecting the reliance of the UMSTMO on constructing maximum spanning trees.

For CA-GrQc dataset, the results of some detection algorithms are not shown (due to memory issues in PercoMCV and all results being very close to 1 in UMSTMO). The Random-Del algorithm outperforms the other community hiding methods in terms of NMI/ONMI in most of the scenarios (refer to Figure 5.14). The 4clqCH algorithm deletes edges that are most commonly shared across 4-cliques. Hence, it starts to disrupt the maximum clique in the graph, and continues with the cliques of lower sizes. The clique distribution within CA-GrQc dataset can be seen from the Figure 5.9. Since the 4clqCH algorithm spends the budget mostly toward dismantling the larger cliques, it struggles to effectively deceive the CPM algorithm with k=4. However, when the k-clique communities with the larger k value (k=7) are identified with the CPM algorithm, it is observed that the 4clqCH algorithm can decrease the ONMI value more.

Table 5.3 illustrates the ONMI results of the 4clqCH, Degree (*122*), and Random-Del algorithms to deceive the CPM (*54*) algorithm on amazon dataset for different values of k ranging from 4 to 7. The 4clqCH algorithm's capability to reduce the ONMI result can be seen even in the lower values of k, showing that it can effectively fool the CPM algorithm. The results of 4clqCH and Degree algorithms against the CPM on youtube dataset for k = 4, 5, and 10 are given in Table 5.4. Even though the maximum clique size in

Table 5.3. ONMI results of 4clqCH, Degree (*122*), and Random-Del algorithms to deceive CPM (*54*) algorithm on amazon dataset.

k-clique communities	Budget	4clqCH	Degree	Random-Del
k=4	3000	**0.9196**	0.9858	0.9902
	4000	**0.9069**	0.9812	0.9868
k=5	3000	**0.8351**	0.9899	0.9794
	4000	**0.7911**	0.9856	0.9722
k=6	3000	**0.0336**	0.9889	0.9541
	4000	**0.4609**	0.9851	0.9402
k=7	3000	**0.0000**	1.0000	0.8889
	4000	**0.2667**	1.0000	0.9111

Table 5.4. ONMI results of 4clqCH and Degree (*122*) algorithms to deceive CPM (*54*) algorithm on youtube dataset.

k-clique communities	Budget	4clqCH	Degree
k=4	3000	**0.7604**	0.8851
k=5	3000	**0.4380**	0.8092
k=10	3000	**0.0862**	0.7449

youtube dataset is k = 17, the 4clqCH algorithm achieves better results than the Degree algorithm starting from the small k values.

Complexity analysis: For four-clique counting, the undirected graph G is transformed into a directed acyclic graph $G^{\rightarrow}$, and the four-clique pattern is decomposed into smaller sub-patterns. These sub-patterns are enumerated within $G^{\rightarrow}$, leveraging the directed nature to effectively reduce the combinatorial expansion of the enumeration process. This approach provides a practical counting. The W_{++} and DD represent the counts of out-wedges and directed diamonds, respectively, as illustrated in Figure 4.5. The computational cost of determining the number of four-cliques per-vertex and per-edge is $O(W_{++}(G^{\rightarrow}) + DD(G^{\rightarrow}))$, requiring an additional $O(m)$ space, where m is the number of edges in the graph. After counting phase, finding the maximum frequency in per-edge counts, getting the edges with the maximum frequency, and finding the victim edge to be deleted takes $O(m)$ time. Then, the common neighbors of the nodes of the chosen edge are determined in $O(d)$ time (d is the maximum node degree). Decreasing the per-edge count of the edges included in the deleted four-cliques is $O(d^2)$. Hence, the complexity after counting is $O(\beta(m + d^2))$, where β is the budget. So, the total complexity of the 4clqCH algorithm is $O(W_{++}(G^{\rightarrow}) + DD(G^{\rightarrow}) + \beta(m + d^2))$.

The LSHA algorithm identifies local structures within the network and subsequently executes a rewiring attack based on these structures. The complexity of detecting local structures is $O(m + dn)$, where n is the number of nodes in the graph. The attack procedure comprises four main steps: 1) selecting the local structures to attack; 2) removing an edge within the low-degree structure; 3) adding an edge between the target node and a node within a high-degree structure; and 4) arranging the local structures. Since step 1 merely selects local structures based on degree, and step 4 applies adjustments to only a small subset of nodes, the primary computational cost lies in steps 2 and 3. The time complexity of step 2 is $O(dm)$ because the vulnerability of each edge within the local structure should

be evaluated. For step 3, the complexity is $O(dn)$, as the entropy calculation represents the most computationally intensive operation. Thus, the time complexity of the attack phase is $O(d(n+m))$. In conclusion, the overall time complexity of the LSHA algorithm is $O(\beta d(n+m))$.

In the Degree algorithm, calculating the degree for all edges is $O(m)$ time. Then, in each iteration, identifying the edge with the maximum degree incurs a time cost of $O(m)$ and updating the degrees of all edges incident to the endpoints is $O(m)$. The time complexity becomes $O(\beta m)$. The space complexity is $O(m+n)$, encompassing the storage of the graph and the additional arrays used for node and edge degrees, adhering to a complexity of $O(m+n)$.

Calculating the betweenness centrality and closeness centrality is computationally expensive since they need all pairs shortest paths. The well-known Brandes' algorithm (*145*) operates at a time complexity of $O(nm)$ for unweighted graphs, and a space complexity of $O(m+n)$. When an edge is removed, recalculating the centrality values for the entire graph can become prohibitively costly, especially if this operation is repeated β times, as dictated by the budget. To address this challenge, new algorithms have been developed to update centrality values in dynamic or evolving graphs (*146*).

The time complexity of Random-Del and Random-Rewire is $O(\beta)$ and $O(\beta n)$, respectively. Their space complexity is $O(m+n)$ for storing the graph.

In summary, since the runtime of 4clqCH is bounded by the numbers of particular patterns, not m and n, its time complexity can be directly compared with that of the Degree and Random algorithms, which have better time complexity. The space complexity of the 4clqCH is $O(m+n)$, being the same as the other community hiding algorithms, except for LSHA, whose space complexity is not reported in the related paper.

Discussion on Findings: Numerous overlapping community detection algorithms leverage cliques or localized information to identify community structures. Attacks against these algorithms aim to disrupt the identified community structures, thereby diminishing their performance. The 4clqCH algorithm focuses on targeting edges mostly found within 4-cliques, prioritizing the disruption of larger cliques by starting with edges in the maximum clique. This approach aligns with efforts in fields like crime detection and prevention, where reducing the size of the maximum clique is critical. Large cliques often signify cohesive substructures, which can serve as potential sources of coordinated activities, such

as terrorist attacks. As such, weakening the cohesiveness of criminal or terrorist networks becomes a crucial objective (*147*).

The effectiveness of the 4clqCH algorithm has been assessed through experiments on various datasets, demonstrating its success in deceiving overlapping community detection algorithms, particularly those based on cliques or seed-based methodologies. Furthermore, the clique distribution within datasets has been analyzed. According to the given budget, when the frequency of large cliques is high, its performance decreases, as observed in the CA-GrQc dataset; however, it remains effective in modifying communities derived from large k-cliques. Additionally, it deceives CPM algorithm on larger datasets, such as amazon and youtube, across different k-values. While the 4clqCH algorithm matches other hiding methods in space complexity, it is outperformed by Degree and Random algorithms in time complexity.

Although significant research has been dedicated to fooling disjoint community detection algorithms, the domain of overlapping community detection has remained largely unexplored when it comes to systematic hiding strategies at a global scale. This study contributes to the field in two ways: first, by evaluating the efficacy of heuristic-based global community hiding algorithms—independent of detection algorithm results—against overlapping algorithms; and second, by proposing and assessing the performance of a new clique-based method.

CHAPTER 6

CONCLUSION AND FUTURE WORK

Within the scope of this thesis, two fields of knowledge hiding techniques for graph data are focused: Subgraph Hiding and Community Hiding. Subgraph hiding aims to protect sensitive subgraphs in transactional graph databases. First, sensitive subgraphs within a database are determined. Then, the database is transformed in a way that prevents the disclosure of these sensitive subgraphs while still preserving the original data as much as possible. That means, the goal is to maintain the utility of the database after publication while ensuring the privacy of the sensitive knowledge it contains. Community hiding, also known as community detection attack, systematically alters the structure of the graph in order to hide communities. With this way, the community membership information of nodes in the graph is preserved.

Various algorithms have been proposed in this thesis for subgraph hiding. These algorithms are EDH, EdgeDegree, Matchings & EdgeDegree, EdgeEntropy and Matchings & EdgeEntropy. All of them choose sensitive graphs with less matches according to the set of sensitive subgraphs for sanitization. Each of them applies different heuristics for local sanitization of a sensitive graph. These algorithms are compared with respect to the execution time and side-effects. All of them have zero hiding failure and do not cause fake patterns. According to the experimental results, the following conclusions can been drawn; When the graphs are sanitized through deleting the selected victim edges, unlike vertex masking, information loss decreases because non-sensitive patterns containing other edges of a vertex may appear after the hiding process. If edges contained in multiple sensitive subgraphs are removed, this causes to remove less edges and therefore the smaller impact on the database. The edge entropy offered to choose the less important edges using the concept of graph entropy has shown similar results to edge degrees. However, since it is based on the subgraph of an edge rather than just its node degrees, it can effectively be used for different tasks.

Following this, a global scale community hiding algorithm, 4clqCH, has been proposed. The 4clqCH performs the modification on the graph by focusing on the four-

cliques and does not use the result of a community detection algorithm. The experimental results have demonstrated the following observation; Since the 4clqCH disturbs the local structures, such as cliques, in the graph, it can provide effective attacks against community detection algorithms that use local information.

The future work can be considered for each problem. For the subgraph hiding problem, transactional databases have been analyzed. But, networks can also be modeled with a single large graph. Subgraph hiding algorithms can be devised to work on a single large graph. For the community hiding problem, hiding algorithms consider only the detection algorithms that focus on the topological structure of networks. But, user characteristics can be set to the network in addition to the relations in the network. That is, hiding techniques can be designed against detection algorithms that consider the attributed networks. The studies in the literature mainly focus on disjoint community detection algorithms. However, overlapping detection algorithms allow more flexible and realistic modeling of systems. Hence, new hiding techniques can be developed to counter overlapping community detection algorithms. For both problems, graphs can be used to represent constantly changing data. Therefore, existing methods can be extended or new solutions can be proposed to work with dynamic graphs.

BIBLIOGRAPHY

1. Potin, L.; Figueiredo, R.; Labatut, V.; Largeron, C. Pattern Mining for Anomaly Detection in Graphs: Application to Fraud in Public Procurement. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, Turin, Italy, Sept. 18–22, 2023; Springer: Cham, Switzerland, 2023; pp 69–87.
2. Mrzic, A.; Meysman, P.; Bittremieux, W.; Moris, P.; Cule, B.; Goethals, B.; Laukens, K. Grasping Frequent Subgraph Mining for Bioinformatics Applications. *BioData Mining* **2018**, 11, 1–24.
3. Wang, L.; Lin, F. V.; Cole, M.; Zhang, Z. Learning Clique Subgraphs in Structural Brain Network Classification with Application to Crystallized Cognition. *NeuroImage* **2021**, 225, 117493.
4. Li, P.-Z.; Huang, L.; Wang, C.-D.; Lai, J.-H. EdMot: An Edge Enhancement Approach for Motif-aware Community Detection. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, Anchorage, AK, USA, Aug. 4–8, 2019; pp 479–487.
5. Ma, J.; Fan, J. Local Optimization for Clique-Based Overlapping Community Detection in Complex Networks. *IEEE Access* **2019**, 8, 5091–5103.
6. Yin, H.; Benson, A. R.; Leskovec, J.; Gleich, D. F. Local Higher-Order Graph Clustering. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Halifax, NS, Canada, Aug. 13–17, 2017; pp 555–564.
7. Javed, M. A.; Younis, M. S.; Latif, S.; Qadir, J.; Baig, A. Community Detection in Networks: A Multidisciplinary Review. *Journal of Network and Computer Applications* **2018**, 108, 87–111.
8. Chen, J.; Chen, L.; Chen, Y.; Zhao, M.; Yu, S.; Xuan, Q.; Yang, X. GA-Based Q-attack on Community Detection. *IEEE Transactions on Computational Social Systems* **2019**, 6, 491–503.
9. Abul, O.; Gökçe, H. Knowledge Hiding from Tree and Graph Databases. *Data & Knowledge Engineering* **2012**, 72, 148–171.

10. Gökçe, H. Ağaç ve Çizge Veritabanlarında Hassas Bilgi Gizleme, MA thesis, TOBB Ekonomi ve Teknoloji Üniversitesi Fen Bilimleri Enstitüsü, 2010.

11. Zhao, J.; Wang, Z.; Cao, J.; Cheong, K. H. A Self-Adaptive Evolutionary Deception Framework for Community Structure. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* **2023**, 53, 4954–4967.

12. Waniek, M.; Michalak, T. P.; Wooldridge, M. J.; Rahwan, T. Hiding Individuals and Communities in a Social Network. *Nature Human Behaviour* **2018**, 2, 139–147.

13. Fionda, V.; Pirro, G. Community Deception or: How to Stop Fearing Community Detection Algorithms. *IEEE Transactions on Knowledge and Data Engineering* **2017**, 30, 660–673.

14. Chen, J.; Chen, Y.; Chen, L.; Zhao, M.; Xuan, Q. Multiscale Evolutionary Perturbation Attack on Community Detection. *IEEE Transactions on Computational Social Systems* **2020**, 8, 62–75.

15. Tekin, L.; Bostanoglu, B. E. Edge Deletion Based Subgraph Hiding. *WSEAS Transactions on Information Science and Applications* **2024**, 21, 333–347.

16. Tekin, L.; Bostanoğlu, B. E. A Qualitative Survey on Community Detection Attack Algorithms. *Symmetry* **2024**, 16, 1272.

17. Liu, D.; Yang, G.; Wang, Y.; Jin, H.; Chen, E. How to Protect Ourselves from Overlapping Community Detection in Social Networks. *IEEE Transactions on Big Data* **2022**, 8, 894–904.

18. Yang, G.; Wang, Y.; Chang, Z.; Liu, D. Overlapping Community Hiding Method Based on Multi-Level Neighborhood Information. *Symmetry* **2022**, 14, 2328.

19. Fayyad, U.; Piatetsky-Shapiro, G.; Smyth, P. From Data Mining to Knowledge Discovery in Databases. *AI Magazine* **1996**, 17, 37–37.

20. Li, L.; Ding, P.; Chen, H.; Wu, X. Frequent Pattern Mining in Big Social Graphs. *IEEE Transactions on Emerging Topics in Computational Intelligence* **2021**, 6, 638–648.

21. Kong, X.; Huang, W.; Tan, Z.; Liu, Y. Molecule Generation by Principal Subgraph Mining and Assembling. *Advances in Neural Information Processing Systems* **2022**, 35, 2550–2563.

22. Queiroz, F. C.; Vargas, A. M.; Oliveira, M. G.; Comarela, G. V.; Silveira, S. A. ppiGReMLIN: A Graph Mining Based Detection of Conserved Structural Arrangements in Protein-Protein Interfaces. *BMC Bioinformatics* **2020**, 21, 1–25.

23. Yigit, Y.; Akram, V. K.; Dagdeviren, O. Breadth-First Search Tree Integrated Vertex Cover Algorithms for Link Monitoring and Routing in Wireless Sensor Networks. *Computer Networks* **2021**, 194, 108144.

24. Han, J.; Pei, J.; Kamber, M. Graph Mining, Social Network Analysis, and Multirelational Data Mining. *Data Mining: Concepts and Techniques* **2006**, 535–589.

25. Inokuchi, A.; Washio, T.; Motoda, H. An Apriori-Based Algorithm for Mining Frequent Substructures from Graph Data. In *Proceedings of the 4th European Conference on Principles of Data Mining and Knowledge Discovery*, Lyon, France, Sept. 13–16, 2000; Springer-Verlag: Berlin, Heidelberg, 2000; pp 13–23.

26. Kuramochi, M.; Karypis, G. An Efficient Algorithm for Discovering Frequent Subgraphs. *IEEE Transactions on Knowledge and Data Engineering* **2004**, 16, 1038–1051.

27. Kuramochi, M.; Karypis, G. Finding Frequent Patterns in a Large Sparse Graph. *Data Mining and Knowledge Discovery* **2005**, 11, 243–271.

28. Huan, J.; Wang, W.; Prins, J. Efficient Mining of Frequent Subgraphs in the Presence of Isomorphism. In *Proceedings of the Third IEEE International Conference on Data Mining*, Melbourne, FL, USA, Nov. 22, 2003; pp 549–552.

29. Yan, X.; Han, J. gspan: Graph-Based Substructure Pattern Mining. In *Proceedings of the 2002 IEEE International Conference on Data Mining*, Maebashi City, Japan, Dec. 9–12, 2002; pp 721–724.

30. Borgelt, C.; Berthold, M. R. Mining Molecular Fragments: Finding Relevant Substructures of Molecules. In *Proceedings of the 2002 IEEE International Conference on Data Mining*, Maebashi City, Japan, Dec. 9–12, 2002; pp 51–58.

31. Yan, X.; Han, J. Closegraph: Mining Closed Frequent Graph Patterns. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Washington, DC, USA, Aug. 24–27, 2003; pp 286–295.

32. Fortunato, S. Community Detection in Graphs. *Physics Reports* **2010**, 486, 75–174.

33. Fortunato, S.; Hric, D. Community Detection in Networks: A User Guide. *Physics Reports* **2016**, 659, 1–44.

34. MacQueen, J. Some Methods for Classification and Analysis of Multivariate Observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*; University of California Press: Berkeley, CA, USA, 1967; Vol. 1, pp 281–297.

35. Hlaoui, A.; Wang, S. A Direct Approach to Graph Clustering. *Neural Networks and Computational Intelligence* **2004**, 4, 158–163.

36. Kernighan, B. W.; Lin, S. An Efficient Heuristic Procedure for Partitioning Graphs. *The Bell System Technical Journal* **1970**, 49, 291–307.

37. Girvan, M.; Newman, M. E. Community Structure in Social and Biological Networks. *Proceedings of the National Academy of Sciences* **2002**, 99, 7821–7826.

38. Newman, M. E. Finding Community Structure in Networks Using the Eigenvectors of Matrices. *Physical Review E—Statistical, Nonlinear, and Soft Matter Physics* **2006**, 74, 036104.

39. Newman, M. E. Spectral Methods for Community Detection and Graph Partitioning. *Physical Review E—Statistical, Nonlinear, and Soft Matter Physics* **2013**, 88, 042822.

40. Higham, D. J.; Kalna, G.; Kibble, M. Spectral Clustering and Its Use in Bioinformatics. *Journal of Computational and Applied Mathematics* **2007**, 204, 25–37.

41. Ruan, J.; Zhang, W. An Efficient Spectral Algorithm for Network Community Discovery and Its Applications to Biological and Social Networks. In *Proceedings of the Seventh IEEE International Conference on Data Mining (ICDM 2007)*, Omaha, NE, USA, Oct. 28–31, 2007; pp 643–648.

42. Brandes, U.; Delling, D.; Gaertler, M.; Gorke, R.; Hoefer, M.; Nikoloski, Z.; Wagner, D. On Modularity Clustering. *IEEE Transactions on Knowledge and Data Engineering* **2007**, 20, 172–188.

43. Chen, M.; Kuzmin, K.; Szymanski, B. K. Community Detection via Maximization of Modularity and Its Variants. *IEEE Transactions on Computational Social Systems* **2014**, 1, 46–65.

44. Newman, M. E. Fast Algorithm for Detecting Community Structure in Networks. *Physical Review E—Statistical, Nonlinear, and Soft Matter Physics* **2004**, 69, 066133.

45. Clauset, A.; Newman, M. E.; Moore, C. Finding Community Structure in Very Large Networks. *Physical Review E* **2004**, 70, 066111.

46. Newman, M. E. Modularity and Community Structure in Networks. *Proceedings of the National Academy of Sciences* **2006**, 103, 8577–8582.

47. Blondel, V. D.; Guillaume, J.-L.; Lambiotte, R.; Lefebvre, E. Fast Unfolding of Communities in Large Networks. *Journal of Statistical Mechanics: Theory and Experiment* **2008**, 2008, P10008.

48. Traag, V. A.; Waltman, L.; Van Eck, N. J. From Louvain to Leiden: Guaranteeing Well-Connected Communities. *Scientific Reports* **2019**, 9, 1–12.

49. Sobolevsky, S.; Campari, R.; Belyi, A.; Ratti, C. General Optimization Technique for High-Quality Community Detection in Complex Networks. *Physical Review E* **2014**, 90, 012811.

50. Pons, P.; Latapy, M. Computing Communities in Large Networks Using Random Walks. In *Proceedings of the Computer and Information Sciences-ISCIS 2005: 20th International Symposium*, Istanbul, Turkey, Oct. 26–28, 2005; Springer: Berlin/Heidelberg, Germany, 2005; pp 284–293.

51. Rosvall, M.; Bergstrom, C. T. Maps of Random Walks on Complex Networks Reveal Community Structure. *Proceedings of the National Academy of Sciences* **2008**, 105, 1118–1123.

52. Reichardt, J.; Bornholdt, S. Statistical Mechanics of Community Detection. *Physical Review E—Statistical, Nonlinear, and Soft Matter Physics* **2006**, 74, 016110.

53. Raghavan, U. N.; Albert, R.; Kumara, S. Near Linear Time Algorithm to Detect Community Structures in Large-Scale Networks. *Physical Review E* **2007**, 76, 036106.

54. Palla, G.; Derényi, I.; Farkas, I.; Vicsek, T. Uncovering the Overlapping Community Structure of Complex Networks in Nature and Society. *Nature* **2005**, 435, 814–818.

55. Kasoro, N.; Kasereka, S.; Mayogha, E.; Vinh, H. T.; Kinganga, J. PercoMCV: A Hybrid Approach of Community Detection in Social Networks. *Procedia Computer Science* **2019**, 151, 45–52.

56. Asmi, K.; Lotfi, D.; El Marraki, M. Overlapping Community Detection based on the Union of All Maximum Spanning Trees. *Library Hi Tech* **2020**, 38, 276–292.

57. Coscia, M.; Rossetti, G.; Giannotti, F.; Pedreschi, D. Demon: A Local-First Discovery Method for Overlapping Communities. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Beijing, China, Aug. 12–16, 2012; pp 615–623.

58. Asmi, K.; Lotfi, D.; Abarda, A. The Greedy Coupled-Seeds Expansion Method for the Overlapping Community Detection in Social Networks. *Computing* **2022**, 104, 295–313.

59. Prat-Pérez, A.; Dominguez-Sal, D.; Larriba-Pey, J.-L. High Quality, Scalable and Parallel Community Detection for Large Real Graphs. In *Proceedings of the 23rd International Conference on World Wide Web*, Seoul, Republic of Korea, Apr. 7–11, 2014; pp 225–236.

60. Fazlali, M.; Moradi, E.; Malazi, H. T. Adaptive Parallel Louvain Community Detection on a Multicore Platform. *Microprocessors and Microsystems* **2017**, 54, 26–34.

61. Al-Andoli, M. N.; Tan, S. C.; Cheah, W. P.; Tan, S. Y. A Review on Community Detection in Large Complex Networks from Conventional to Deep Learning Methods: A Call for the Use of Parallel Meta-Heuristic Algorithms. *IEEE Access* **2021**, 9, 96501–96527.

62. Waniek, M.; Woźnica, J.; Zhou, K.; Vorobeychik, Y.; Michalak, T. P.; Rahwan, T. Hiding from Centrality Measures: A Stackelberg Game Perspective. *IEEE Transactions on Knowledge and Data Engineering* **2023**, 35, 10058–10071.

63. Waniek, M.; Zhou, K.; Vorobeychik, Y.; Moro, E.; Michalak, T. P.; Rahwan, T. How to Hide One's Relationships from Link Prediction Algorithms. *Scientific Reports* **2019**, 9, 12208.

64. Dey, P.; Medya, S. Manipulating Node Similarity Measures in Networks. *arXiv* **2019**, arXiv:1910.11529.

65. Waniek, M.; Holme, P.; Cebrian, M.; Rahwan, T. Social Diffusion Sources Can Escape Detection. *Iscience* **2022**, 25, 104956.

66. Oliveira, S. R.; Zaiane, O. R. Privacy Preserving Frequent Itemset Mining. In *Proceedings of the IEEE ICDM Workshop on Privacy, Security, and Data Mining*, Maebashi City, Japan, Dec. 2002; pp 43–54.

67. Gkoulalas-Divanis, A.; Loukides, G. Revisiting Sequential Pattern Hiding to Enhance Utility. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, San Diego, CA, USA, Aug. 21–24, 2011; pp 1316–1324.

68. Newman, M. E.; Girvan, M. Finding and Evaluating Community Structure in Networks. *Physical Review E* **2004**, 69, 026113.

69. Newman, M. E. Analysis of Weighted Networks. *Physical Review E* **2004**, 70, 056131.

70. Danon, L.; Diaz-Guilera, A.; Duch, J.; Arenas, A. Comparing Community Structure Identification. *Journal of Statistical Mechanics: Theory and Experiment* **2005**, 2005, P09008.

71. Hubert, L.; Arabie, P. Comparing Partitions. *Journal of Classification* **1985**, 2, 193–218.

72. McDaid, A. F.; Greene, D.; Hurley, N. Normalized Mutual Information to Evaluate Overlapping Community Finding Algorithms. *arXiv* **2011**, arXiv:1110.2515.

73. Chen, J.; Wu, Y.; Xu, X.; Chen, Y.; Zheng, H.; Xuan, Q. Fast Gradient Attack on Network Embedding. *arXiv* **2018**, arXiv:1809.02797.

74. Bernini, A.; Silvestri, F.; Tolomei, G. Community Membership Hiding as Counterfactual Graph Search via Deep Reinforcement Learning. *arXiv* **2023**, arXiv:2310.08909.

75. Liu, D.; Jia, R.; Liu, X.; Zhang, W. A Unified Framework of Community Hiding using Symmetric Nonnegative Matrix Factorization. *Information Sciences* **2024**, 663, 120235.

76. Nagaraja, S. The Impact of Unlinkability on Adversarial Community Detection: Effects and Countermeasures. In *Proceedings of the International Symposium on Privacy Enhancing Technologies Symposium*, Berlin, Germany, July 21–23, 2010; Springer: Berlin/Heidelberg, Germany, 2010; pp 253–272.

77. Mittal, S.; Sengupta, D.; Chakraborty, T. Hide and Seek: Outwitting Community Detection Algorithms. *IEEE Transactions on Computational Social Systems* **2021**, 8, 799–808.

78. Meilă, M. Comparing Clusterings by the Variation of Information. In *Proceedings of the Learning Theory and Kernel Machines: 16th Annual Conference on Learning Theory and 7th Kernel Workshop, COLT/Kernel*, Washington, DC, USA, Aug. 24–27, 2003; Springer: Berlin/Heidelberg, Germany, 2003; pp 173–187.

79. Van Dongen, S. Performance Criteria for Graph Clustering and Markov Cluster Experiments. *Report-Information Systems* **2000**, 1–36.

80. Liu, Y.; Liu, J.; Zhang, Z.; Zhu, L.; Li, A. REM: From Structural Entropy to Community Structure Deception. *Advances in Neural Information Processing Systems* **2019**, 32.

81. Liu, X.; Fu, L.; Wang, X.; Hopcroft, J. E. Prohico: A Probabilistic Framework to Hide Communities in Large Networks. In *Proceedings of the IEEE INFOCOM 2021-IEEE Conference on Computer Communications*, Vancouver, BC, Canada, May 10–13, 2021; pp 1–10.

82. Kumari, S.; Yadav, R. J.; Namasudra, S.; Hsu, C.-H. Intelligent Deception Techniques against Adversarial Attack on the Industrial System. *International Journal of Intelligent Systems* **2021**, 36, 2412–2437.

83. Liu, D.; Chang, Z.; Yang, G.; Chen, E. Community Hiding using a Graph Autoencoder. *Knowledge-Based Systems* **2022**, 253, 109495.

84. Liu, D.; Chang, Z.; Yang, G.; Chen, E. Hiding Ourselves from Community Detection through Genetic Algorithms. *Information Sciences* **2022**, 614, 123–137.

85. Shannon, C. E. A Mathematical Theory of Communication. *The Bell System Technical Journal* **1948**, 27, 379–423.

86. Cao, S.; Dehmer, M.; Shi, Y. Extremality of Degree-Based Graph Entropies. *Information Sciences* **2014**, 278, 22–33.

87. Qiao, T.; Shan, W.; Zhou, C. How to Identify the Most Powerful Node in Complex Networks? A Novel Entropy Centrality Approach. *Entropy* **2017**, 19, 614.

88. Sen, B.; Chu, S.-H.; Parhi, K. K. Ranking Regions, Edges and Classifying Tasks in Functional Brain Graphs by Sub-graph Entropy. *Scientific Reports* **2019**, 9, 1–20.

89. Sun, X.; Yu, P. S. A Border-Based Approach for Hiding Sensitive Frequent Itemsets. In *Fifth IEEE International Conference on Data Mining (ICDM'05)*, Houston, TX, USA, Nov. 27–30, 2005; pp 426–433.

90. Moustakides, G. V.; Verykios, V. S. A Maxmin Approach for Hiding Frequent Itemsets. *Data & Knowledge Engineering* **2008**, 65, 75–89.

91. Sharma, S.; Toshniwal, D. MR-I MaxMin-Scalable Two-Phase Border Based Knowledge Hiding Technique using MapReduce. *Future Generation Computer Systems* **2020**, 109, 538–550.

92. Krasadakis, P.; Futia, G.; Verykios, V. S.; Sakkopoulos, E. Graph based Hiding of Sensitive Knowledge. In *2023 IEEE 35th International Conference on Tools with Artificial Intelligence (ICTAI)*, Atlanta, GA, USA, Nov. 6–8, 2023; pp 199–203.

93. Menon, S.; Sarkar, S.; Mukherjee, S. Maximizing Accuracy of Shared Databases when Concealing Sensitive Patterns. *Information Systems Research* **2005**, 16, 256–270.

94. Gkoulalas-Divanis, A.; Verykios, V. S. Hiding Sensitive Knowledge without Side Effects. *Knowledge and Information Systems* **2009**, 20, 263–299.

95. Verykios, V. S.; Stavropoulos, E. C.; Krasadakis, P.; Sakkopoulos, E. Frequent Itemset Hiding Revisited: Pushing Hiding Constraints into Mining. *Applied Intelligence* **2022**, 52, 2539–2555.

96. Guo, Y. Reconstruction-Based Association Rule Hiding. In *Proceedings of SIGMOD2007 PhD Workshop on Innovative Database Research*, Beijing, China, June 10, 2007; pp 51–56.

97. Li, S.; Mu, N.; Le, J.; Liao, X. Privacy Preserving Frequent Itemset Mining: Maximizing Data Utility based on Database Reconstruction. *Computers & Security* **2019**, 84, 17–34.

98. Lin, C.-W.; Hong, T.-P.; Yang, K.-T.; Wang, S.-L. The GA-Based Algorithms for Optimizing Hiding Sensitive Itemsets through Transaction Deletion. *Applied Intelligence* **2015**, 42, 210–230.

99. Lin, J. C.-W.; Liu, Q.; Fournier-Viger, P.; Hong, T.-P.; Voznak, M.; Zhan, J. A Sanitization Approach for Hiding Sensitive Itemsets based on Particle Swarm Optimization. *Engineering Applications of Artificial Intelligence* **2016**, 53, 1–18.

100. Telikani, A.; Gandomi, A. H.; Shahbahrami, A.; Dehkordi, M. N. Privacy-Preserving in Association Rule Mining using an Improved Discrete Binary Artificial Bee Colony. *Expert Systems with Applications* **2020**, 144, 113097.

101. Dasseni, E.; Verykios, V. S.; Elmagarmid, A. K.; Bertino, E. Hiding Association Rules by Using Confidence and Support. In *International Workshop on Information Hiding*, Pittsburgh, PA, USA, Apr. 25–27, 2001; pp 369–383.

102. Oliveira, S. R.; Zaiane, O. R. Protecting Sensitive Knowledge by Data Sanitization. In *Third IEEE International Conference on Data Mining*, Melbourne, FL, USA, Nov. 19–22, 2003; pp 613–616.

103. Amiri, A. Dare to Share: Protecting Sensitive Knowledge with Data Sanitization. *Decision Support Systems* **2007**, 43, 181–191.

104. Hong, T.-P.; Lin, C.-W.; Yang, K.-T.; Wang, S.-L. Using TF-IDF to Hide Sensitive Itemsets. *Applied Intelligence* **2013**, 38, 502–510.

105. Cheng, P.; Roddick, J. F.; Chu, S.-C.; Lin, C.-W. Privacy Preservation through a Greedy, Distortion-Based Rule-Hiding Method. *Applied Intelligence* **2016**, 44, 295–306.

106. Saygin, Y.; Verykios, V. S.; Clifton, C. Using Unknowns to Prevent Discovery of Association Rules. *ACM Sigmod Record* **2001**, 30, 45–54.

107. Wang, S.-L.; Jafari, A. Using Unknowns for Hiding Sensitive Predictive Association Rules. In *IRI-2005 IEEE International Conference on Information Reuse and Integration*, Las Vegas, NV, USA, Aug. 15–17, 2005; pp 223–228.

108. Zachary, W. W. An Information Flow Model for Conflict and Fission in Small Groups. *Journal of Anthropological Research* **1977**, 33, 452–473.

109. Mnih, V.; Badia, A. P.; Mirza, M.; Graves, A.; Lillicrap, T.; Harley, T.; Silver, D.; Kavukcuoglu, K. Asynchronous Methods for Deep Reinforcement Learning. In *Proceedings of the International Conference on Machine Learning*, New York, NY, USA, June 19–24, 2016; pp 1928–1937.

110. Fionda, V.; Pirró, G. Community Deception in Weighted Networks. In *Proceedings of the 2021 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, Virtual Event Netherlands, Nov. 8–11, 2021; pp 278–282.

111. Fionda, V.; Madi, S. A.; Pirrò, G. Community Deception: From Undirected to Directed Networks. *Social Network Analysis and Mining* **2022**, 12, 74.

112. Fionda, V.; Pirrò, G. Community Deception in Attributed Networks. *IEEE Transactions on Computational Social Systems* **2022**, 11, 228–237.

113. Madi, S. A.; Pirrò, G. Community Deception in Directed Influence Networks. *Social Network Analysis and Mining* **2023**, 13, 122.

114. Chen, X.; Jiang, Z.; Li, H.; Ma, J.; Philip, S. Y. Community Hiding by Link Perturbation in Social Networks. *IEEE Transactions on Computational Social Systems* **2021**, 8, 704–715.

115. Chakraborty, T.; Srinivasan, S.; Ganguly, N.; Mukherjee, A.; Bhowmick, S. Permanence and Community Structure in Complex Networks. *ACM Transactions on Knowledge Discovery from Data (TKDD)* **2016**, 11, 1–34.

116. Nallusamy, K.; Easwarakumar, K. PERMDEC: Community Deception in Weighted Networks using Permanence. *Computing* **2024**, 106, 353–370.

117. Zhang, C.; Fu, L.; Ding, J.; Cao, X.; Long, F.; Wang, X.; Zhou, L.; Zhang, J.; Zhou, C. Community Deception in Large Networks: Through the Lens of Laplacian Spectrum. *IEEE Transactions on Computational Social Systems* **2023**, 11, 2057–2069.

118. Madi, S. A.; Pirrò, G. Node-Centric Community Deception Based on Safeness. *IEEE Transactions on Computational Social Systems* **2023**, 11, 2955–2965.

119. Pirrò, G. Community Deception From a Node-Centric Perspective. *IEEE Transactions on Network Science and Engineering* **2023**, 11, 969–981.

120. Chang, Z.; Liang, J.; Ma, S.; Liu, D. Community Hiding: Completely Escape from Community Detection. *Information Sciences* **2024**, 672, 120665.

121. Zhao, J.; Wang, Z.; Yu, D.; Cao, J.; Cheong, K. H. Swarm Intelligence for Protecting Sensitive Identities in Complex Networks. *Chaos, Solitons & Fractals* **2024**, 182, 114831.

122. Wang, S.; Liu, J. Constructing Robust Community Structure against Edge-Based Attacks. *IEEE Systems Journal* **2018**, 13, 582–592.

123. Yu, S.; Zheng, J.; Chen, J.; Xuan, Q.; Zhang, Q. Unsupervised Euclidean Distance Attack on Network Embedding. In *Proceedings of the 2020 IEEE Fifth International Conference on Data Science in Cyberspace (DSC)*, Hong Kong, China, July 27–29, 2020; pp 71–77.

124. Magelinski, T.; Bartulovic, M.; Carley, K. M. Measuring Node Contribution to Community Structure with Modularity Vitality. *IEEE Transactions on Network Science and Engineering* **2021**, 8, 707–723.

125. Kipf, T. N.; Welling, M. Variational Graph Auto-Encoders. *arXiv* **2016**, arXiv:1611.07308.

126. Yang, H.; Chen, L.; Cheng, F.; Qiu, J.; Zhang, L. LSHA: A Local Structure-Based Community Detection Attack Heuristic Approach. *IEEE Transactions on Computational Social Systems* **2023**, 11, 2966–2978.

127. Zhao, J.; Cheong, K. H. Obfuscating Community Structure in Complex Network with Evolutionary Divide-and-Conquer Strategy. *IEEE Transactions on Evolutionary Computation* **2023**, 27, 1926–1940.

128. Yang, S.; Chen, B.; Zhu, G. EPCG: An Elite Population Co-evolutionary Genetic Algorithm for Global Community Deception. In *Proceedings of the 7th International Conference on Control Engineering and Artificial Intelligence*, Sanya, China, Jan. 28–30, 2023; pp 66–71.

129. Wang, X.; Li, J.; Guan, Y.; Yuan, J.; Tao, H.; Zhang, S. Enhancing Community Deception based on Graph Autoencoder and Genetic Algorithm. In *Proceedings of the 2023 IEEE 9th International Conference on Computer and Communications (ICCC)*, Chengdu, China, Dec. 8–11, 2023; pp 742–746.

130. Gupta, S. K.; Singh, D. P.; Choudhary, J. A Review of Clique-Based Overlapping Community Detection Algorithms. *Knowledge and Information Systems* **2022**, 64, 2023–2058.

131. Kumpula, J. M.; Kivelä, M.; Kaski, K.; Saramäki, J. Sequential Algorithm for Fast Clique Percolation. *Physical Review E* **2008**, 78, 026109.

132. Maity, S.; Rath, S. K. Extended Clique Percolation Method to Detect Overlapping Community Structure. In *2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI 2014)*, Delhi, India, Sept. 24–27, 2014; pp 31–37.

133. Zhang, X.; Wang, C.; Su, Y.; Pan, L.; Zhang, H.-F. A Fast Overlapping Community Detection Algorithm based on Weak Cliques for Large-Scale Networks. *IEEE Transactions on Computational Social Systems* **2017**, 4, 218–230.

134. Gupta, S. K.; Singh, D. P. CBLA: A Clique Based Louvain Algorithm for Detecting Overlapping Community. *Procedia Computer Science* **2023**, 218, 2201–2209.

135. Pinar, A.; Seshadhri, C.; Vishal, V. Escape: Efficiently Counting All 5-vertex Subgraphs. In *Proceedings of the 26th International Conference on World Wide Web*, Perth, Australia, Apr. 3–7, 2017; pp 1431–1440.

136. Srinivasan, A.; King, R. D.; Muggleton, S. H.; Sternberg, M. J. The Predictive Toxicology Evaluation Challenge. In *Proceedings of the 15th International Joint Conference on Artifical Intelligence*, Nagoya, Japan, Aug. 23–29, 1997; pp 4–9.

137. Cantador, I.; Brusilovsky, P.; Kuflik, T. Second Workshop on Information Heterogeneity and Fusion in Recommender Systems (HetRec2011). In *Proceedings of the Fifth ACM Conference on Recommender Systems*, Chicago, Illinois, USA, Oct. 23–27, 2011; pp 387–388.

138. Nguyen, D.; Luo, W.; Nguyen, T. D.; Venkatesh, S.; Phung, D. Learning Graph Representation via Frequent Subgraphs. In *Proceedings of the 2018 SIAM International Conference on Data Mining*, San Diego, California, USA, May 3–5, 2018; pp 306–314.

139. Cordella, L. P.; Foggia, P.; Sansone, C.; Vento, M. A (Sub) Graph Isomorphism Algorithm for Matching Large Graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **2004**, 26, 1367–1372.

140. Bröhl, T.; Lehnertz, K. A Straightforward Edge Centrality Concept derived from Generalizing Degree and Strength. *Scientific Reports* **2022**, 12, 4407.

141. Choumane, A.; Awada, A.; Harkous, A. Core Expansion: A New Community Detection Algorithm based on Neighborhood Overlap. *Social Network Analysis and Mining* **2020**, 10, 1–11.

142. Gregori, E.; Lenzini, L.; Mainardi, S. Parallel K-Clique Community Detection on Large-Scale Networks. *IEEE Transactions on Parallel and Distributed Systems* **2012**, 24, 1651–1660.

143. Rossi, R. A.; Ahmed, N. K. The Network Data Repository with Interactive Graph Analytics and Visualization. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, Austin, TX, USA, Jan. 25–30, 2015; pp 4292–4293.

144. Leskovec, J.; Krevl, A. SNAP Datasets: Stanford Large Network Dataset Collection, `http://snap.stanford.edu/data`, 2014.

145. Brandes, U. A Faster Algorithm for Betweenness Centrality. *Journal of Mathematical Sociology* **2001**, 25, 163–177.

146. Shukla, K.; Regunta, S. C.; Tondomker, S. H.; Kothapalli, K. Efficient Parallel Algorithms for Betweenness-and Closeness-Centrality in Dynamic Graphs. In *Proceedings of the 34th ACM International Conference on Supercomputing*, Barcelona, Spain, June 29–July 2, 2020; pp 1–12.

147. Furini, F.; Ljubić, I.; Martin, S.; San Segundo, P. The Maximum Clique Interdiction Problem. *European Journal of Operational Research* **2019**, 277, 112–127.

Leyla Tekin

Vita

Academic Experience

2017–2024 Research/Teaching Assistant, Department of Computer Engineering, *İzmir Institute of Technology*

Education

2015–2018 MSc in Computer Engineering, *İzmir Institute of Technology*

2010–2015 BSc in Computer Engineering, *İzmir Institute of Technology*

Publications

2024 Leyla Tekin and Belgin Ergenç Bostanoğlu. A qualitative survey on community detection attack algorithms. *Symmetry*, 16(10):1272, 2024.

2024 Leyla Tekin and Belgin Ergenç Bostanoğlu. Edge deletion based subgraph hiding. *WSEAS Transactions on Information Science and Applications*, 21:333-347, 2024.

2018 Leyla Tekin. Analysing the encrypted search algorithms on encrypted data. Master's thesis, İzmir Institute of Technology, 2018.

2018 Leyla Tekin, Hüseyin Güven Özgür, Burcu Sayin, Arzum Karataş, Pelin Şenkula, Emre İrtem, and Serap Şahin. EDU-VOTING: An educational homomorphic e-voting system. In *International Conference on Advanced Technologies, Computer Engineering and Science (ICATCES'18)*, pages 114-119, 2018.

2017 Leyla Tekin and Serap Şahin. Implementation and evaluation of improved secure index scheme using standard and counting bloom filters. *International Journal of Information Security Science*, 6(4):46-56, 2017.

2017 Leyla Tekin and Serap Şahin. Implementation and evaluation of improved secure index scheme using standard and counting bloom filters. In *10th International Conference on Information Security and Cryptology (ISCTurkey 2017)*, pages 167-174, 2017.

Awards & Honors

2015 The Rector's List (3rd Highest-Ranked Student Award), *İzmir Institute of Technology*

2015 2nd Highest-Ranked Student Award in Computer Engineering Graduates, *İzmir Institute of Technology*