Manipulating Structural Graph Clustering

Wentao Li[♮], Min Gao[§], Dong Wen[‡], Hongwei Zhou[§], Cai Ke[§], and Lu Qin[♮]

[♮]*AAII, FEIT, University of Technology Sydney;* [‡]*The University of New South Wales;* [§]*Chongqing University*

[♮]{wentao.li, lu.qin}@uts.edu.au; [‡]dong.wen@unsw.edu.au; [§]{gaomin, zhouhw, cqukecai}@cqu.edu.cn

***Abstract*—Structural graph clustering (SCAN) is a popular clustering technique. Using the concept of ϵ-neighborhood, SCAN defines the core vertices that uniquely determine the clusters of a graph. Most existing studies assume that the graph processed by SCAN contains no controlled edges. Few studies, however, have focused on manipulating SCAN by injecting edges. Manipulation of SCAN can be used to assess its robustness and lay the groundwork for developing robust clustering algorithms. To fill this gap and considering the importance of the ϵ-neighborhood for SCAN, we propose a problem, denoted as MN, for manipulating SCAN. The MN problem aims to maximize the ϵ-neighborhood of the target vertex by inserting some edges. On the theoretical side, we prove that the MN problem is both NP-hard and APX-hard, and also is non-submodular and non-monotonic. On the algorithmic side, we design an algorithm by focusing on how to select vertices to join ϵ-neighborhood and thus avoid enumerating edges to report a solution. As a result, our algorithm bypasses the non-monotonicity nature of the MN problem. Extensive experiments on real-world graphs show that our algorithm can effectively solve the proposed MN problem.**

***Index Terms*—Structural graph clustering, Manipulation, Graph, Algorithm, NP-Hard**

I. Introduction

Graphs (or networks) are widely used to represent schemaless data [1], such as data from social networks, collaborative networks, and biological networks [2], [3]. As a fundamental task in understanding graph structure, graph clustering has received increasing attention [4]. Graph clustering aims to place the vertices of a graph into clusters such that vertices in the same cluster have dense connectivity and vertices in different clusters have sparse connectivity [4].

Graph clustering has many applications [5]. For example, it can be used to find groups with similar research interests in collaborative networks [4], or customers with similar purchasing preferences in e-commerce networks [6]. Due to the numerous applications, many graph clustering methods have been proposed [2], [7]–[9]. Among these, structural graph clustering (SCAN) is a popular method [1], [4], [10]–[18] for identifying clusters as well as hubs and outliers.

A. Structural Graph Clustering

SCAN relies on the concept of the ϵ-neighborhood, and employs two parameters ϵ and μ [12]. Given a graph $G(V, E)$ with vertices V and edges E, the **neighborhood** $N_G[u]$ of a vertex u includes the neighbors v adjacent to u as well as u itself, i.e., $N_G[u] = \{v|(u, v) \in E\} \cup \{u\}$. SCAN defines the similarity $\sigma_G(u, v)$ between two vertices u, v according to the Cosine similarity of their neighborhoods (see Definition 2). If $\sigma_G(u, v) \geq \epsilon$, then vertices u and v are similar to each other, where ϵ is the parameter used by SCAN. The **ϵ-neighborhood** $N^{\epsilon}_G[u]$ of u is defined as the vertices in its neighborhood that are similar to it, i.e., $N^{\epsilon}_G[u] = \{v|v \in N_G[u], \sigma_G(u, v) \geq \epsilon\}$.

The ϵ-neighborhood (size) is crucial for SCAN: a vertex is identified as the **core** if its ϵ-neighborhood size is not smaller than a parameter μ; the core vertices uniquely determine the graph's clusters [4]. Meanwhile, a vertex not contained in any cluster is a **hub** if vertices in its neighborhood belong to at least two clusters; otherwise, it is an **outlier**. In this manner, SCAN assigns roles to all vertices in the graph. For example, for the graph in Fig. 1, SCAN identifies two clusters, one hub, and one outlier.

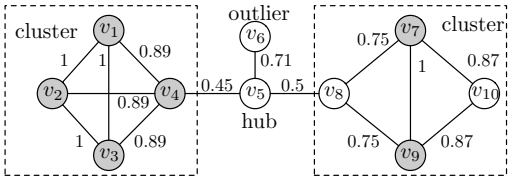


Fig. 1: SCAN under $\epsilon = 0.7$, $\mu = 4$. v_4 is a core, because $N^{\epsilon}_G(v_4) = \{v_1, v_2, v_3, v_4\}$ contains four vertices; v_5 is a hub, because its two neighbors v_4 and v_8 are in different clusters; v_6 is an outlier.

B. Motivations

SCAN is useful in many applications [10], [11], but existing research assumes that SCAN works with "clean" graphs in which vertices and edges accurately reflect the information. However, due to the openness of network platforms [19], [20], controlled edges [21], [22] can be injected into the graph. For example, in social networks such as Facebook [23], controlled links (i.e., edges) can be created with other accounts [21], [22]. There has been little research to understand the behavior of SCAN when edges are allowed to be injected.

To fill the gap, this paper is concerned with whether the results of SCAN can be manipulated by inserting edges. Identifying potential manipulations of the algorithm and assessing their impact [24]–[26] is commonly used in the algorithmic robustness analysis. When it comes to SCAN, it is necessary to evaluate its robustness via manipulations due to the widespread use of SCAN.

We attempt to manipulate SCAN from the perspective of varying ϵ-neighborhood size, considering the importance of ϵ-neighborhood size for SCAN. For example, if an outlier vertex successfully expands its ϵ-neighborhood size to at least the parameter μ, SCAN incorrectly identifies this vertex as a core. Other aspects of SCAN manipulation will be investigated in future studies. Specifically, we propose the **maximum ϵ-neighborhood problem** (MN), that is, given a target vertex t of G and a budget k, we intend to insert (at most) k edges into G so that the ϵ-neighborhood size of t increases the most.

Solving the MN problem can be used to evaluate the robustness of SCAN. Indeed, we can infer SCAN's robustness by examining the amount of improvement in the size of the target vertex's ϵ-neighborhood after inserting some edges: if the improvement amount after insertion is large, we can conclude that SCAN is vulnerable to manipulation by attacks. Therefore, designing an effective algorithm to solve the MN problem (and thus obtain the improvement amount accurately) is necessary. We hope that our investigation of the MN problem will raise awareness for further research on designing more robust clustering algorithms.

C. Contributions

We try to solve the MN problem because of its importance. We investigate the theoretical difficulty of this problem and devise an effective method to solve it.

Theoretical Results. The MN problem is challenging because we may need to check all combinations containing (up to) k edges. The solution is the edge combination that maximizes the target vertex t's ϵ-neighborhood size after the insertion. Indeed, we prove that the MN problem is NP-hard (see Theorem 1). This implies that finding the exact solution in polynomial time is futile. Approximation algorithms can be used to speed up, but we show that the MN problem is also APX-hard (see Theorem 2). This indicates that obtaining solutions with good approximation ratios is also difficult. The challenge of the MN problem also lies in the fact that this problem is neither submodular *nor monotonic* (see Theorem 3). This means that increasing the budget (i.e., the number of inserted edges) may decrease t's ϵ-neighborhood size.

Algorithmic Results. To tame the challenge of the MN problem, rather than enumerating combinations of (up to k) edges, we consider selecting vertices to be added. That is, given a budge k, we want to add as many vertices as possible to t's ϵ-neighborhood (through inserting edges) to maximize its size. Recall that t's ϵ-neighborhood contains only vertices whose similarity to t is not less than ϵ. To make the above idea work, we need to solve two subproblems. (1) The local promotion problem, i.e., how to raise the similarity between some vertex v and t (to at least ϵ) so that v joins t's ϵ-neighborhood. (2) The global selection problem, i.e., how to select as many vertices for local promotion as possible, given a budget k.

We propose a degree-based technique to solve the first subproblem (local promotion), which ensures that some vertex v is added to t's ϵ-neighborhood at minimal cost under certain conditions; we propose a cost-based technique to solve the second subproblem (global selection), which ensures that a large number of vertices are added to t's ϵ-neighborhood to expand its size for a given budget. By resolving these two subproblems, we obtain the algorithm to tackle MN. Our algorithm solves the problem in the unit of the vertex, ensuring that increasing the budget k can only increase t's ϵ-neighborhood size, thus bypassing the non-monotonicity nature of the MN problem.

The contributions of this paper are summarized as follows.

- *Formalization of the* MN *problem* (Section II). We propose to manipulate SCAN by varying the ϵ-neighborhood size. To this end, we formally define the MN problem. To the best of our knowledge, the manipulation of SCAN has not been studied in previous work.
- *Hardness analysis of* MN (Section III). We show that the MN problem is challenging by proving that it is NP-hard and APX-hard. Moreover, we find that this problem is non-submodular and non-monotonic. These analyses show that designing an effective algorithm to solve this problem is non-trivial.
- *Algorithm design for* MN (Section IV). We solve the MN problem by identifying its two key subproblems: local promotion and global selection. By solving these subproblems, we obtain the algorithm to deal with the MN problem. Furthermore, similarity measures are crucial in many machine learning tasks such as recommendation systems and link prediction [27]. Therefore, investigating the similarity promotion between two vertices (i.e., local promotion) may be of independent interest for these machine learning tasks.
- *Extensive experimental studies* (Section V). To validate the effectiveness of the devised algorithm, we conducted experiments on various real-world graphs. The experimental results show that for a given budget, our algorithm can effectively expand the ϵ-neighborhood size of the target vertex. Furthermore, the ϵ-neighborhood size grows as the budget increases.

D. Related Work

Clustering Manipulation. Algorithm manipulation is widely used in the algorithmic robustness analysis [24]–[26], and it serves as the foundation for designing robust algorithms [28]–[30]. The literature [31], [32] provided a survey of algorithm manipulation. Here, we mainly discuss the manipulation of clustering algorithms. Crussell et al. [33] manipulated the density-based clustering method DBSCAN [34] by merging two separated clusters. Chhabra et al. [35] proposed a black-box manipulation for k-means clustering. Manipulations for single-linkage hierarchical clustering [36], spectral clustering [37], and lowest ID clustering [38] are also available. However, to the best of our knowledge, there are no studies on manipulating SCAN.

Structural Graph Clustering. Structural graph clustering (SCAN) was first proposed in [12], which will be described in the following section. Since then, many studies have been done to improve the efficiency of SCAN. Shiokawa et al. [1] devised a new data structure to accelerate SCAN. Chang et al. [4] further improved the efficiency by using a new two-step paradigm. A MapReduce-based SCAN algorithm was designed in [13], and a multi-core based algorithm was devised in [15]. An approximate algorithm using edge sampling was given in [14].

Another line of studies focuses on the parameters ϵ and μ required for SCAN. The methods proposed in [16], [17] automatically find the best ϵ, making these methods free of the parameter ϵ. Wen et al. [18] created the GS$*$-Index, which stores clustering results under any ϵ and μ. Tseng et al. [10] provided a parallel construction of the GS$*$-Index.

II. Preliminary

We first introduce SCAN in Section II-A, and then formally define the problem under study in Section II-B.

A. Structural Graph Clustering

We focus on an undirected unweighted graph $G(V, E)$, which includes $n = |V|$ vertices and $m = |E|$ edges. For each vertex $u \in V$, we define the **neighbors** of u, denoted by $N_G(u)$, as the vertices v adjacent to u, i.e., $N_G(u) = \{v \in V | (u, v) \in E\}$. The **path** $p_G = (v_1, v_2, \cdots, v_l)$ from v_1 to v_l is a sequence of vertices in G, where $(v_i, v_{i+1}) \in E$ and $i \in [1, l-1]$.

Definition 1 (Neighborhood)**.** The neighborhood of a vertex $u \in V$ in G, denoted by $N_G[u]$, is defined as $N_G[u] = N_G(u) \cup \{u\} = \{v \in V | (u, v) \in E\} \cup \{u\}$.

Example 1. Consider the graph G in Fig. 1. The neighbors of v_7 are $\{v_8, v_9, v_{10}\}$. The neighborhood $N_G[v_7]$ of v_7 is $\{v_7, v_8, v_9, v_{10}\}$.

The **degree** of a vertex u, denoted by $d_G[u]$, is the size of $N_G[u]$, i.e., $d_G[u] = |N_G[u]|$. For two vertices u and v, we denote the vertices shared by their neighborhoods as $N_G[u, v] = \{w | w \in \{N_G[u] \cap N_G[v]\}\}$, and its size by $d_G[u, v] = |N_G[u, v]|$. We denote the vertices belonging to $N_G[u]$ but not to $N_G[v]$ as $N_G[u - v]$, i.e., $N_G[u - v] = \{w | w \in \{N_G[u] \setminus N_G[v]\}\}$.

Definition 2 (Similarity)**.** The similarity between two vertices u and v in G, denoted by $\sigma_G(u, v)$, is the Cosine similarity of their neighborhoods, i.e., the number $d_G[u, v]$ of shared vertices, normalized by the geometric mean $\sqrt{d_G[u] \times d_G[v]}$ of their degrees. That is, $\sigma_G(u, v) = \frac{d_G[u,v]}{\sqrt{d_G[u] \times d_G[v]}}$.

Example 2. Consider the graph G in Fig. 1. The degree of v_7 is $d_G[v_7] = 4$, and the degree of v_8 is $d_G[v_8] = 4$. For v_7 and v_8, the vertices shared by their neighborhoods are $N_G[v_7, v_8] = \{v_7, v_8, v_9\}$, and $d_G[v_7, v_8] = 3$. The similarity between v_7 and v_8 is $\sigma_G(v_7, v_8) = \frac{3}{\sqrt{4\times4}} = 0.75$. The vertices in $N_G[v_7]$ but not in $N_G[v_8]$ is $N_G[v_7 - v_8] = \{v_{10}\}$.

Definition 3 (ϵ-Neighborhood)**.** Given a parameter ϵ ($0 < \epsilon \leq 1$), the ϵ-neighborhood of a vertex u in G, denoted by $N^\epsilon_G[u]$, is defined as the subset of $N_G[u]$, where each vertex v satisfies $\sigma_G(u, v) \geq \epsilon$. That is, $N^\epsilon_G[u] = \{v \in N_G[u] | \sigma_G(u, v) \geq \epsilon\}$.

Definition 4 (Core)**.** Given an integer μ ($\mu \geq 2$), a vertex u is a core of G if $|N^\epsilon_G[u]| \geq \mu$.

Definition 5 (Structural Reachability)**.** Given a core vertex u and another vertex v in G, v is structurally reachable from u if there exists a path $(v_1, v_2, \cdots, v_l)$ ($l \geq 2$) such that, i) $v_1 = u, v_l = v$; ii) for all $1 \leq i \leq l - 1$, v_i is core, and $v_{i+1} \in N^\epsilon_G[v_i]$.

Example 3. Consider the graph G in Fig. 1. With parameters $\epsilon = 0.7, \mu = 4$, v_8 is in $N^\epsilon_G[v_7]$ since $\sigma_G(v_7, v_8) = 0.75 \geq \epsilon$. The ϵ-neighborhood $N^\epsilon_G[v_7]$ of v_7 is $\{v_7, v_8, v_9, v_{10}\}$, and v_7 is a core. The ϵ-neighborhood $N^\epsilon_G[v_5]$ of v_5 is $\{v_5, v_6\}$, and v_5 is not a core. v_8 is structurally reachable from v_7 since there is a path (v_7, v_8) between them and $v_8 \in N^\epsilon_G[v_7]$.

SCAN *Algorithm.* We are now ready to describe how SCAN clusters the graph.

1) Compute the similarity (by Definition 2) of all vertex pairs (u, v), where $(u, v) \in E$.
2) Compute $N^\epsilon_G[u]$ for $\forall u \in V$ (by Definition 3) to obtain all core vertices $C = \{u | |N^\epsilon_G[u]| \geq \epsilon\}$ (by Definition 4).
3) Initialize all core vertices in C as unvisited, and then take out one unvisited vertex $c \in C$ at a time to find a cluster.
   a) Assign all vertices structurally reachable (by Definition 5) from unvisited vertex $c \in C$ to the same **cluster**.
   b) Set the vertices in this cluster to be visited.
   c) Stop if all vertices in C are marked as visited; otherwise, continue with step 3-a) to find other clusters.

With the above, SCAN finds all the clusters in the graph. For the vertices not in any cluster, we classify them into hubs or outliers, according to their neighborhoods.

Definition 6 (Hub and Outlier)**.** Given a vertex u that does not belong to any cluster, u is a hub if vertices in its neighborhood belong to at least two clusters. Otherwise, u is an outlier.

Example 4. Consider the graph G in Fig. 1. (1) We compute (and label on edges) the similarity of all vertex pairs. (2) We obtain the ϵ-neighborhood of each vertex thus determining the core vertices $C = \{v_1, v_2, v_3, v_4, v_7, v_9\}$. (3) We take an unvisited core vertex $v_1 \in C$ and find all vertices $\{v_1, v_2, v_3, v_4\}$ that are structurally reachable by v_1 as the first cluster. Then we take out another unvisited vertex $v_7 \in C$ and thus find the second cluster $\{v_7, v_8, v_9, v_{10}\}$. At this time, all vertices in C are marked as visited, and the clustering ends. For vertices not in any cluster, we identify v_5 as a hub and v_6 as an outlier.

B. Problem Formulation

The ϵ-neighborhood size is critical for SCAN: a vertex is a core when its ϵ-neighborhood size is not smaller than the parameter μ; core vertices uniquely determine clusters in a graph [4]. Due to the importance of the ϵ-neighborhood size, we consider manipulating SCAN by varying the ϵ-neighborhood size of the target vertex.

Formally, given the graph $G(V, E)$, we try to insert some edges $\overline{E} \subseteq \{V^2 \setminus E\}$ to obtain a modified graph $G'(V', E')$, where $V' = V$ and $E' = E \cup \overline{E}$. For a target vertex $t \in V$: in G, t's ϵ-neighborhood is denoted as $N^\epsilon_G[t]$; in G', t's ϵ-neighborhood is denoted as $N^\epsilon_{G'}[t]$. Then, given the number of edges allowed to be inserted, i.e., the budget k, the **maximum ϵ-neighborhood** (MN) problem is defined as follows.

- **Input**: graph G, parameter ϵ, budge k, and target vertex t;
- **Output**: (at most) k edges $\overline{E} \subseteq \{V^2 \setminus E\}$ to modify $G(V, E)$ to $G'(V, E \cup \overline{E})$ so that $|N^\epsilon_{G'}[t]|$ is maximized.

The optimization objective of the MN problem is to maximize t's ϵ-neighborhood size $|N^\epsilon_{G'}[t]|$ in the modified graph G'. This objective is equivalent to maximizing the amount of improvement $\Delta[t] = |N^\epsilon_{G'}[t]| - |N^\epsilon_G[t]|$, i.e., the amount of change in t's ϵ-neighborhood size of t before and after edge insertion. *We will use these two objectives indiscriminately*.

Example 5. Consider the graph G in Fig. 1. Given $\epsilon = 0.8$, budget $k = 1$, and target vertex $t = v_6$, one solution to the MN problem is the edge set $\overline{E} = \{(v_6, v_8)\}$. Before insertion, $N^\epsilon_G[v_6] = \{v_6\}$; after insertion, $N^\epsilon_{G'}[v_6] = \{v_5, v_6\}$. The improvement amount (objective value) is $\Delta[v_6] = 1$.

Remark. To modify a graph G, we only consider inserting edges in G because inserting a vertex is equivalent to adding edges adjacent to that vertex. Also, we have simplified the MN problem to facilitate discussion. We still have the following options to investigate for further work. (1) All edges are assumed to be successfully inserted; however, inserting edges should be possible with only a certain probability. (2) The insertion cost is assumed to be the same for all edges; however, different edges may have different costs. (3) SCAN is only manipulated by changing the ϵ-neighborhood size; however, SCAN can be manipulated in other ways, such as merging two clusters.

III. Theoretical Results

In this section, we examine the hardness of the MN problem. Section III-A shows that the MN problem is NP-hard, and Section III-B demonstrates that it is APX-hard, non-submodular and non-monotonic.

A. NP-Hardness

We prove that the MN problem is NP-hard, implying that trying to solve the problem exactly in polynomial time is futile.

Theorem 1. *The* MN *problem is NP-hard.*

Proof. We reduce from the maximum coverage (MC) problem [39], which is also NP-hard and defined as follows.

- **Input**: universal set $\mathbb{P} = \{e_1, e_2, \cdots, e_p\}$ with p elements e_l ($1 \leq l \leq p$); $\mathbb{Q} = \{S_1, S_2, \cdots, S_q\}$ with q sets, where each set $S_j \in \mathbb{Q}$ is a non-empty subset of $\mathbb{P}$ ($1 \leq j \leq q$); budget k.
- **Output**: $\overline{\mathbb{Q}}$ containing (at most) k sets from $\mathbb{Q}$ that together cover the most elements in $\mathbb{P}$.

Next, we create an instance of the MN problem in Fig. 2. The goal is to insert (up to) k edges in G to form G' so that $|N^\epsilon_{G'}[t]|$

of the target vertex t is maximized. Here, $\epsilon = \frac{3}{d_G[t]+k}$. The graph G in Fig. 2 consists of five components:

- target vertex t that is connected to vertices in sets A, B, P;
- path $A = \{a_1, a_2, \cdots, a_w\}$ with w vertices, where w is set to ensure that t is the maximum degree node in G;
- set $P = \{e_1, e_2, \cdots, e_p\}$with p vertices, where each $e_l \in P$ $(1 \le l \le p)$ corresponds to an element $e_l \in \mathbb{P}$ in MC;
- h equivalent vertex sets Q_i $(1 \le i \le h)$, where each set $Q_i = \{S_{i1}, S_{i2}, \cdots, S_{iq}\}$ contains q vertices. Each vertex $S_{ij} \in Q_i$ $(1 \le j \le q)$ corresponds to a set $S_j \in \mathbb{Q}$. We connect $S_{ij} \in Q_i$ to some vertex $e_l \in P$ if $e_l \in S_j$ in the MC problem. We set $h > \lceil \frac{4}{9} \frac{(d_G[t]+k)^2}{d_G[t]} \rceil$;
- set $B = \{B_1, B_2, \cdots, B_q\}$ of q vertices, where each $B_j \in B$ $(1 \le j \le q)$ connects to vertex $S_{ij} \in Q_i$, for all $1 \le i \le h$. For example, B_1 connects to $\{S_{11}, S_{2,1}, \cdots, S_{h,1}\}$.

In G, $N^\epsilon_G[t] = \{t, a_1, a_2, \cdots, a_w\}$ since

1) for each $a \in A$, $\sigma_G(t,a) \ge \frac{3}{\sqrt{d_G[t] \times 4}} \ge \epsilon$.
2) each $e_l \in P$ is not in $N^\epsilon_G[t]$ $(\sigma_G(t,e_l) < \frac{2}{\sqrt{h \times d_G[t]}} < \epsilon)$ as
   a) $d_G[t,e_l] = 2$ as $N_G[t] \cap N_G[e_l] = \{t, e_l\}$;
   b) $d_G[e_l] \ge h+1$ as e_l is connected to at least one vertex S_{ij} in all h groups Q_i.
3) each $B_j \in B$ is not in $N^\epsilon_G[t]$ $(\sigma_G(t,B_j) = \frac{2}{\sqrt{(h+2) \times d_G[t]}} < \epsilon)$ since $d_G[t,B_j] = 2$ and $d_G[B_j] = h+2$.

Solution to MC $\Rightarrow$ *solution to* MN. Assume that k sets $\overline{\mathbb{Q}}$ are chosen from $\mathbb{Q}$ for MC, which together cover the maximum elements $\overline{\mathbb{P}}$ of $\mathbb{P}$. $\overline{\mathbb{Q}}$ corresponds to vertices $\overline{Q_1}$ in group Q_1, and $\overline{\mathbb{P}}$ corresponds to vertices $\overline{P}$ in P. Then, we connect t to each vertex of $\overline{Q_1}$ to form G'. Also, we denote $\overline{B}$ as the vertices in B that are connected to vertices in $\overline{Q_1}$. In G', $N^\epsilon_{G'}[t] = N^\epsilon_G[t] \cup \overline{P} \cup \overline{Q_1} \cup \overline{B}$ since

1) for each $e_l \in \overline{P}$, $d_G[t,e_l] \ge 3$ as at least one vertex in $\overline{Q_1}$ is a new neighbor between t and e_l;
2) for each $S_{1j} \in \overline{Q_1}$, $d_G[t,S_{1j}] \ge 3$ as at least one vertex in $\overline{B}$ is a new neighbor between t and S_{1j};
3) for each $B_j \in \overline{B}$, $d_G[t,B_j] = 3$ as $S_{1j} \in \overline{Q_1}$ is a new neighbor between t and B_j.

t is still of maximum degree in G', so $\sigma_{G'}(t,e_r) \ge \epsilon$, $\sigma_{G'}(t,S_{1j}) \ge \epsilon$, and $\sigma_{G'}(t,B_j) \ge \epsilon$.

Solution to MN $\Rightarrow$ *solution to* MC. First, the solution to the MN problem must be to connect t to some vertices S_{ij} in groups Q_i, since this makes at least 3 vertices lie in $N^\epsilon_{G'}[t]$ (1 for S_{ij}, 1 for neighbor $B_j \in B$, and 1 for neighbor in P). The other ways can increase the size by at most 2 (e.g., inserting edges between vertices in P or B). Second, when the selected k vertices are scattered in different groups Q_i, we can gather them in Q_1 while improving the same size, since all groups Q_i are equivalent. In Q_1, the selected vertices $\overline{Q_1}$ (resp., sets of $\overline{\mathbb{Q}}$) are actually the solution to MC, since they together cover the maximum vertices in P (resp., elements of $\mathbb{P}$) such that these vertices are in $N^\epsilon_{G'}[t]$. □

B. APX-Hardness and Beyond

The MN problem is also hard to approximate.

Theorem 2. *The* MN *problem cannot be approximated in polynomial time within a ratio of* $(1 - \frac{1}{e} + \eta)$*, for* $\forall \eta > 0$*, unless P=NP.*

Proof. If OPT(MC) is the optimal objective value for MC and APX(MC) is an approximate, OPT(MC) $\ge$ APX(MC). If MN has an optimal objective value OPT(MN) $= |N^\epsilon_{G'}[t]| - |N^\epsilon_G[t]|$, this value is $|\overline{P} \cup \overline{Q_1} \cup \overline{B}|$ according to the proof of Theorem 1. OPT(MN) = OPT(MC) + $2k$ because $|\overline{Q_1}| = |\overline{B}| = k$ and $|\overline{P}|$ = OPT(MC). Similarly, the approximate value of MM is APX(MN) = APX(MC) + $2k$. If MN has an approximate ratio $\gamma > 1 - \frac{1}{e}$, i.e., APX(MN) $\ge \gamma$OPT(MN), then γOPT(MC) + $2\gamma k = \gamma$OPT(MN) $\le$ APX(MN) = APX(MC) + $2k$. Hence, $\frac{\text{APX(MC)}}{\text{OPT(MC)}} \ge \gamma + \frac{2(\gamma-1)k}{\text{OPT(MC)}}$. This implies MC has an approximate ratio at least γ, which contradicts [40]. □

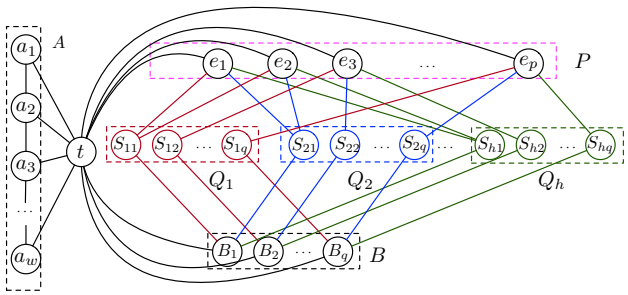


Fig. 2: Proof of NP-Hardness

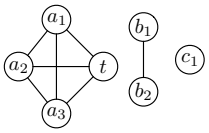


Fig. 3: Proof of Non-Submodular and Non-Monotonic

We then show that the MN problem is neither submodular nor monotonic. In conjunction with Theorem 2, it implies the difficulty of designing (greedy) algorithms with good approximate ratios. This motivates us to investigate a practical method to solve the problem in the next section.

Theorem 3. *Let* $\Delta[t] = |N^\epsilon_{G'}[t]| - |N^\epsilon_G[t]|$ *be t's improvement amount, we have* $\Delta[t]$ *is non-submodular and non-monotonic.*

Proof. We denote $\Delta_S[t]$ as the improvement when edges S are inserted into G, i.e., $\Delta_S[t] = |N^\epsilon_{G'(V,E\cup S)}[t]| - |N^\epsilon_{G(V,E)}[t]|$.

Non-Submodular. Suppose $\Delta[t]$ is submodular, then $\Delta_A[t] + \Delta_B[t] \ge \Delta_{A\cup B}[t] + \Delta_{A\cap B}[t]$ for any edge sets A and B. Consider the graph G in Fig. 3. Let $A = \{(t,b_1)\}$, $B = \{(t,b_2)\}$ and $\epsilon = \frac{1}{\sqrt{2}}$, then, $N^\epsilon_G[t] = \{t, a_1, a_2, a_3\}$ in G.

1) $\Delta_A[t] = 0$: after inserting $A = \{(t,b_1)\}$ to form G', $b_1 \notin N^\epsilon_{G'}[t]$ as $\sigma_{G'}(t,b_1) = \frac{2}{\sqrt{5\times 3}} < \epsilon$;
2) $\Delta_B[t] = 0$: after inserting $B = \{(t,b_2)\}$ to form G', $b_2 \notin N^\epsilon_{G'}[t]$;
3) $\Delta_{A\cup B}[t] = 2$: after inserting A and B to form G', both b_1 and b_2 are in $N^\epsilon_{G'}[t]$ as $\sigma_{G'}(t,b_1) = \sigma_{G'}(t,b_2) = \frac{3}{\sqrt{6\times 3}} = \epsilon$;
4) $\Delta_{A\cap B}[t] = 0$ since $G' = G$.

As a result, $\Delta_A[t] + \Delta_B[t] < \Delta_{A\cup B}[t] + \Delta_{A\cap B}[t]$, contradiction.

Non-Monotonic. Suppose $\Delta[t]$ is monotonic, then $\Delta_{S'}[t] \ge \Delta_S[t]$ for $\forall S \subseteq S'$. When edges $S = A \cup B = \{(t,b_1),(t,b_2)\}$ are inserted to form G', $\Delta_S[t] = 2$; when edges $S' = S \cup \{(b_1,c_1)\}$ are inserted to form G', $\Delta_{S'}[t] = 1$ as $\sigma_{G'}(t,b_1) = \frac{3}{\sqrt{6\times 4}} < \epsilon$, contradiction. □

IV. Algorithmic Results

Section IV-A identifies two key subproblems of the MN problem, which are then solved in Sections IV-B and IV-C, respectively. The overall algorithm for MN is given in Section IV-D. The proofs are moved to the Appendix for clarity.

A. Overview

An intuitive way to solve the MN problem is to enumerate all combinations of (up to) k edges from $\{V^2 \setminus E\}$, and compare the improvement in the target vertex t's ϵ-neighborhood size before and after inserting these edges. The edge combination that produces the greatest improvement is the solution to MN. This way, however, necessitates checking a large number of edge combinations, rendering it intractable. Also, Theorem 3 states that the MN problem is non-monotonic. This is a negative result because increasing the budget k may lead to a smaller ϵ-neighborhood size improvement of t.

Can we design a method to bypass the non-monotonicity nature of the MN problem? We will give an affirmative answer to this question. Our main idea is to approach the problem by selecting vertices rather than enumerating edges. That is, we find the inserted edges by determining which vertices can be added to t's ϵ-neighborhood. Note that t's ϵ-neighborhood contains only vertices whose similarity to t is not less than ϵ. The key to solving the MN problem then lies in determining how to raise the similarity between a vertex v and t to at least ϵ, so that v joins t's ϵ-neighborhood (see Subproblem 1) and how to select as many vertices as possible to promote the similarity when given a budget k (see Subproblem 2).

Subproblem 1 (Local Promotion)**.** Given a target vertex t and another vertex $v \notin N_G^\epsilon[t]$, insert the least edges to modify G to G', such that $\sigma_{G'}(t, v) \geq \epsilon$ in G'.

Subproblem 2 (Global Selection)**.** Given a target vertex t and a budget k, select vertices from $V \setminus N_G^\epsilon[t]$ into $N_{G'}^\epsilon[t]$ in some priority order, to maximize $|N_{G'}^\epsilon[t]|$ in G'.

Subproblem 1 is to use the least edges to increase the similarity between v and t to ϵ so that v is in t's ϵ-neighborhood. Subproblem 2 is to maximize t's ϵ-neighborhood size for a given budget k, by choosing as many vertices v as possible to increase the similarity (by calling Subproblem 1). Since we determine the inserted edges in the unit of the vertex, increasing the budget can only increase the number of added vertices (i.e., the improvement of t's ϵ-neighborhood size). This bypasses the non-monotonicity nature of MN. We will solve these two subproblems in the sequel.

B. Local Promotion

This section addresses Subproblem 1, i.e., local similarity promotion for vertices t and v. We first introduce the strategies for similarity promotion in Section IV-B-1, then discuss how to choose the preferred strategy in Section IV-B-2, and finally give other factors to consider in Sections IV-B-3 and IV-B-4.

B-1. Promotion Strategies

According to the definition of ϵ-neighborhood (see Definition 3), two conditions are required for a vertex v to be a member of the target vertex t's ϵ-neighborhood. The first condition is that the similarity between v and t is not less than ϵ, and the second condition is that v belongs to t's neighborhood $N_G[t]$. We assume that $v \in N_G[t]$, and the extension on $v \notin N_G[t]$ will be given in Section IV-B-4.

Then, the only condition to let (neighbor) v join t's ϵ-neighborhood is to raise the similarity $\sigma_G(t, v)$ between them to at least ϵ. By the definition, $\sigma_G(t, v) = \frac{d_G[t,v]}{\sqrt{d_G[t] \times d_G[v]}}$. Since inserting edges in G does not decrease the degree of any vertex

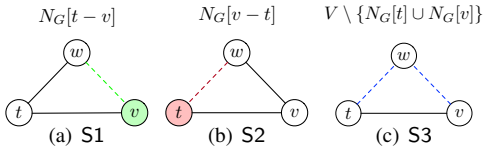


Fig. 4: Insertion Strategies

(the denominator does not decrease), the *only* way to raise the similarity is to increase the number $d_G[t, v]$ (numerator): let t and v share more neighbors. Based on this consideration, all (possible) **edge insertion strategies** are as follows.

- S1. connect v to $w \in N_G[t - v] = N_G[t] \setminus N_G[v]$ (Fig. 4(a));
- S2. connect t to $w \in N_G[v - t] = N_G[v] \setminus N_G[t]$ (Fig. 4(b));
- S3. connect t, v to $w \in \{V \setminus \{N_G[v] \cup N_G[t]\}\}$ (Fig. 4(c)).

Example 6. Consider the graph G in Fig. 1. We describe three strategies to improve the similarity between the target vertex $t = v_5$ and $v = v_8$. (1) S1: we connect $v = v_8$ with $v_6 \in N_G[t-v] = \{v_4, v_6\}$. (2) S2: we connect $t = v_5$ with $v_7 \in N_G[v - t] = \{v_7, v_9\}$. (3) S3: we connect both $t = v_5$ and $v = v_8$ to $v_3 \in \{V \setminus \{N_G[v] \cup N_G[t]\}\}$.

B-2. Degree-Based Strategy Determination

We present three strategies (S1, S2, and S3) to show how to insert edges for similarity promotion, and the next question is how to choose the **preferred strategy**. The preferred strategy for vertices t and v is to use a small number of edges so that their similarity reaches ϵ. Note that the similarity between t and v improves monotonically with the number of inserted edges (see Lemmas 1-2).

With this in mind, to determine the preferred strategy, we can pre-inject l edges to see how much the similarity has improved for each possible strategy. The preferred strategy is the one that yields a large similarity value after injection. Based on the above idea, this section presents the degree-based strategy determination technique to identify the preferred strategy.

Validity of S1 *and* S2. We first show that by inserting edges between v and $N_G[t-v]$ (via S1) in G to form G', the similarity between t and v will increase in G'. This implies that S1 is valid for similarity promotion.

Lemma 1. *If l edges $\overline{E} \subseteq \{(v, w)|w \in N_G[t - v]\}$ are inserted in G to form $G'(V, E \cup \overline{E})$ (by* S1*), $\sigma_{G'}(t, v) > \sigma_G(t, v)$, where $l \leq |N_G[t - v]|$.*

By iteratively using Lemma 1, we can prove that the similarity value increases monotonically as the number of inserted edges increases. With similar logic, we show that if edges are inserted between the target vertex t and $N_G[v - t]$ (by S2) in G to form G', the similarity between t and v will be increased in G'. This implies that S2 is also valid.

Lemma 2. *If l edges $\overline{E} \subseteq \{(t, w)|w \in N_G[v - t]\}$ are inserted in G to form $G'(V, E \cup \overline{E})$ (by* S2*), $\sigma_{G'}(t, v) > \sigma_G(t, v)$, where $l \leq |N_G[v - t]|$.*

Example 7. Consider the graph G in Fig. 1. We want to increase the similarity between $t = v_5$ and $v = v_8$. In G, $\sigma_G(v_5, v_8) = \frac{2}{\sqrt{4 \times 4}} = 0.5$. By S1, we connect v_8 with $v_6 \in N_G[v_5 - v_8]$ to form G', and $\sigma_{G'}(v_5, v_8) = \frac{3}{\sqrt{4 \times 5}} = 0.67 > \sigma_G(v_5, v_8)$. By S2, we connect v_5 with $v_7 \in N_G[v_8 - v_5]$ to form G', and $\sigma_{G'}(v_5, v_8) = 0.67 > \sigma_G(v_5, v_8)$.

Hybrid Strategy Using S1 *and* S2. We use S1 to insert edges between v and $N_G[t-v]$, or S2 to insert edges between t and $N_G[v-t]$. We can also use a hybrid of S1 and S2, i.e., simultaneously select some vertices in $N_G[t-v]$ to connect to v and some vertices in $N_G[v-t]$ to connect to t. However, the maximum similarity value is obtained when this **hybrid strategy** degenerates to S1 or S2.

Lemma 3. *If l edges $\overline{E} \subseteq \{(v,w)|w \in N_G[t-v]\} \cup \{(t,w)|w \in N_G[v-t]\}$ are inserted in G to form $G'(V, E \cup \overline{E})$, $\sigma_{G'}(t,v)$ maximizes when either $\overline{E} \subseteq \{(v,w)|w \in N_G[t-v]\}$ (by* S1*) or $\overline{E} \subseteq \{(t,w)|w \in N_G[v-t]\}$ (by* S2*), where $l \leq \min\{|N_G[t-v]|, |N_G[v-t]|\}$.*

Example 8. Consider the graph G in Fig. 1, where $t = v_5$ and $v = v_8$. When we use the hybrid strategy to insert two edges (v_8, v_6) (by S1) and (v_5, v_7) (by S2), we obtain $\sigma_{G'}(v_5, v_8) = \frac{4}{\sqrt{5\times5}} = 0.8$. When we use S1 to insert two edges between v_8 and $N_G[v_5 - v_8]$ or we use S2 to insert two edges between v_5 and $N_G[v_8 - v_5]$, we get $\sigma_{G'}(v_5, v_8) = \frac{4}{\sqrt{4\times6}} = 0.82$. This shows that the maximum similarity value of the hybrid strategy arises when S1 or S2 is used alone.

Priority of S1 *and* S2. Lemma 3 shows that the best performance of the hybrid strategy occurs when S1 or S2 is used alone. However, there is still a question: we cannot decide when it is suitable to use S1 and when to use S2. Lemma 4 provides the answer.

Lemma 4. *Suppose $\sigma^{s1}_{G'}(t,v)$ is obtained by choosing l edges from $\{(v,w)|w \in N_G[t-v]\}$ (by* S1*) and $\sigma^{s2}_{G'}(t,v)$ is obtained by choosing l edges from $\{(t,w)|w \in N_G[v-t]\}$ (by* S2*), where $l \leq \min\{|N_G[t-v]|, |N_G[v-t]|\}$, then*

$$\begin{cases} \sigma^{s1}_{G'}(t,v) > \sigma^{s2}_{G'}(t,v), & \text{if } d_G[t] < d_G[v]; \\ \sigma^{s1}_{G'}(t,v) < \sigma^{s2}_{G'}(t,v), & \text{if } d_G[t] > d_G[v]. \end{cases}$$

From Lemma 4, it follows that S1 is preferable when $d_G[v] > d_G[t]$ while S2 is preferable when $d_G[v] < d_G[t]$.

Example 9. Consider the graph G in Fig. 1. Suppose we want to increase the similarity between $t = v_5$ and $v = v_4$. Note that $d_G[v_5] = 4 < d_G[v_4] = 5$. When we apply S1 to insert an edge between v_4 and $N_G[v_5 - v_4]$, we obtain $\sigma^{S1}_{G'}(v_4, v_5) = \frac{3}{\sqrt{6\times4}} = 0.62$. When we apply S2 to insert an edge between v_5 and $N_G[v_4 - v_5]$, we obtain $\sigma^{S2}_{G'}(v_4, v_5) = \frac{3}{\sqrt{5\times5}} = 0.6$. Thus, S1 outperforms S2 when $d_G[v_5] < d_G[v_4]$.

Priority of S3. Strategy S3 connects both t and v to the vertex $w \in \{V \setminus \{N[v] \cup N[t]\}\}$. In this case, we need to insert 2 edges to create a common neighbor for t and v. We show that S3 is not as effective as S1 or S2.

Lemma 5. *Suppose $\sigma^{s3}_{G'}(t,v)$ is obtained by inserting l edges from $\{(t,w),(v,w)|w \in \{V \setminus \{N[v] \cup N[t]\}\}\}$ (by* S3*), $\sigma^{s1}_{G'}(t,v)$ is obtained by inserting l edges from $\{(v,w)|w \in N_G[t-v]\}$ (by* S1*), $\sigma^{s2}_{G'}(t,v)$ is obtained by inserting l edges from $\{(t,w)|w \in N_G[v-t]\}$ (by* S2*), where $l \leq \min\{|N_G[t-v]|, |N_G[v-t]|\}$, then, $\sigma^{s3}_{G'}(t,v) < \min\{\sigma^{s1}_{G'}(t,v), \sigma^{s2}_{G'}(t,v)\}$.*

Example 10. Consider the graph G in Fig. 1, where $t = v_5$ and $v = v_8$. When we use S3 to insert two edges (v_5, v_3) and (v_8, v_3), we obtain $\sigma^{S3}_{G'}(v_5, v_8) = \frac{3}{\sqrt{5\times5}} = 0.6$. When we use S1 to insert two edges between v_8 and $N_G[v_5 - v_8]$ or we use S2 to insert two edges between v_5 and $N_G[v_8 - v_5]$, we obtain $\sigma_{G'}(v_5, v_8) = \frac{4}{\sqrt{4\times6}} = 0.82$. This shows that S3 is less effective than S1 or S2.

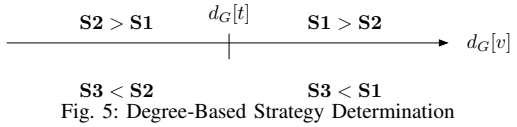


Fig. 5: Degree-Based Strategy Determination

Remark. S3 can be combined with either S1 or S2 to create a hybrid strategy. However, because S1 and S2 outperform S3 in terms of similarity promotion, the best performance of a hybrid strategy occurs when S3 is not used. As a result, hybrid strategies that include S3 are not considered.

Degree-Based Technique. We propose the degree-based technique for strategy determination (Fig. 5). Assume that the degree of the target vertex t is $d_G[t]$ in G, and vertices in G are divided into two groups, one greater than $d_G[t]$ and one less than $d_G[t]$.

- According to Lemma 4, S1 outperforms S2 for vertices with degree greater than $d_G[t]$, while S2 outperforms S1 for vertices with degree less than $d_G[t]$;
- According to Lemma 3, the hybrid strategy performs best when it degenerates to S1 or S2.
- According to Lemma 5, S3 is no better than S1 and S2.

Thus, to choose the preferred strategy for raising the similarity between vertices t and v, we avoid using S3, and use only S1 and S2. Specifically,

- If $d_G[t] < d_G[v]$, we use S1;
- If $d_G[t] > d_G[v]$, we use S2;
- If $d_G[t] = d_G[v]$, we use S1 because using S2 has a side effect that will be discussed in Section IV-B-3.

In this way, when all strategies are applicable, the degree-based technique guarantees that the chosen strategy is optimal, i.e., the similarity between t and v can be increased to ϵ by inserting the minimum number of edges.

B-3. Side Effect of S2 *and* S3

We plan to increase the similarity between the target vertex t and another vertex $v \in V$. However, using strategies S2 and S3 will simultaneously increase the degree of t: using S2, t will connect to vertices $T \subseteq N_G[v-t]$; using S3, t will connect to vertices $T \subseteq \{V \setminus \{N_G[v] \cup N_G[t]\}$. This will have a **side effect** on other vertices in $N_G[t]$, as their similarity with t may decrease due to the increase in the degree of t.

Lemma 6. *When t connects to vertices in T by* S2 *or* S3*, then $\sigma_{G'}(t,u) < \sigma_G(t,u)$, for vertices $\{u \in N_G[t] | N_G[u] \cap T = \emptyset\}$.*

The seriousness of the side effect is that it may cause the similarity between t and vertices u in $N^{\epsilon}_G[t]$ to drop below ϵ, thus putting u outside $N^{\epsilon}_G[t]$. In other words, the side effect can make the size of t's ϵ-neighborhood size decrease after inserting edges, violating our requirement for monotonicity.

Example 11. Consider the graph G in Fig. 1, where $\epsilon = 0.7$. Suppose we want to boost the similarity between $t = v_5$ and $v = v_8$, we use S2 to connect v_5 to $v_7 \in N_G[v_8 - v_5]$. For $v_6 \in N^{\epsilon}_G[v_5]$, as the degree of v_5 increase by 1 in G', $\sigma_{G'}(v_5, v_6) = 0.63 < \sigma_G(v_5, v_6) = 0.71$. This has a side effect on v_6 since its similarity with v_5 decreases to below ϵ.

Side Effect Handling. To handle the side effect on $u \in N_G^\epsilon[t]$, we need to let u *rejoin* $N_G^\epsilon[t]$. Thus, we need to repair the similarity between t and u. But an ensuing problem is that in order to repair their similarity, we may further increase the degree of t, causing this side effect cascades. Fortunately, we show that the degree of t can be kept constant during the similarity repair process, which can be done by connecting u to vertices in T.

Lemma 7. *Suppose $l = |T| = d_{G'}[t] - d_G[t]$ edges are inserted between t and vertices in T, for $\forall u \in N_G^\epsilon[t]$, $\sigma_{G'}(t,u) \geq \epsilon$ when inserting $\leq l$ edges between u and T.*

Example 12. Following the previous example, the similarity between $v = v_6$ and $t = v_5$ drops below $\epsilon = 0.7$, so v_6 leaves v_5's ϵ-neighborhood. To eliminate the side effect, we connect v_6 to $T = \{v_7\}$. After this operation, $\sigma_{G'}(v_5, v_6) = \frac{3}{\sqrt{5\times 3}} = 0.77$, thereby rejoining v_6 to $N_{G'}^\epsilon[v_5]$. During this process, the degree of $t = v_5$ remains constant.

Algorithm. Suppose the degree of t increases because of connecting to vertices in T. If the similarity between t and $u \in N_G^\epsilon[t]$ is below ϵ, we connect u to a subset T' of T to eliminate the side effect. Lemma 7 guarantees that this treatment definitely restores their similarity to at least ϵ (if $T = T'$). Specifically, in Algorithm 1, we first set the result set $\overline{E}$ as an empty set (Line 1). Then, the modified graph G' is obtained by connecting t to vertices in T (Line 2). Next, for each $u \in N_G^\epsilon[t]$, if the similarity between t and u is lower than ϵ, we progressively[1] add vertices from T to T' until the similarity recovers to at least ϵ (Line 4-5). We connect u to T' and insert the corresponding edges to $\overline{E}$ (Line 6).

Algorithm 1: Side Effect Handling

Input: graph $G(V, E)$, target t, vertices T
Output: edges $\overline{E}$
1 $\overline{E} \leftarrow \emptyset$;
2 $G' \leftarrow G(V, E \cup \{(t,w)|w \in T\})$;
3 **for** *each vertex* $u \in N_G^\epsilon[t]$ **do**
4 **if** $\sigma_{G'}(t,u) < \epsilon$ **then**
5 add vertices from T to T' until $\sigma_{G'}(t,u) \geq \epsilon$;
6 $\overline{E} \leftarrow \overline{E} \cup \{(u,w)|w \in T'\}$;
7 **return** $\overline{E}$;

Theorem 4. *The time cost[2] of Algorithm 1 is $O(|T| \cdot |N_G^\epsilon[t]| \cdot d_{\max})$, where $d_{\max}$ is the maximum degree of vertices in G.*

B-4. Extension to Non-Neighborhood

To let v join the target vertex t's ϵ-neighborhood, the condition $v \in N_G[t]$ is necessary because $N_G^\epsilon[t] \subseteq N_G[t]$. When $v \notin N_G[t]$, we need to do a preprocessing by connecting t and v to make v become t's neighbor. However, this procedure increases the degree of t by 1 (t connects to $T = \{v\}$), which leads to the side effect, as introduced in Lemma 6. Hence, we also need to use Algorithm 1 to handle this side effect.

Algorithm. Algorithm 2 gives the preprocessing for $v \notin N_G[t]$. We first call Algorithm 1 to fix the side effect caused by connecting t to v (Line 1). Then, the edge (t,v) is added to $\overline{E}$ (Line 2). With this preprocessing, we can assume that all vertices v are in t's neighborhood.

[1]We add vertices in T in the order of their non-increasing vertex IDs.
[2]The factor $|T|$ can be removed by applying Equations 1-2 in Section IV-C.

Algorithm 2: Preprocessing

Input: graph $G(V, E)$, target t, $v \notin N_G[t]$
Output: edges $\overline{E}$
1 $\overline{E} \leftarrow$ Algorithm 1 by setting $T = \{v\}$;
2 $\overline{E} \leftarrow \overline{E} \cup \{(t,v)\}$;
3 **return** $\overline{E}$;

Theorem 5. *The time cost of Algorithm 2 is $O(|N_G^\epsilon[t]| \cdot d_{\max})$, where $d_{\max}$ is the maximum degree of vertices in G.*

Example 13. Consider the graph G in Fig. 1. Suppose we want to improve the similarity between $t = v_5$ and $v = v_1$, we first connect v_5 to v_1 as preprocessing. However, the increase in the degree of v_5 causes the similarity between v_5 and $v_6 \in N_G^\epsilon[v_5]$ to drop to $\sigma_{G'}(v_5, v_6) = \frac{2}{\sqrt{5\times 2}} = 0.63$. We apply the side effect handling technique to connect v_6 to $T = \{v_1\}$. After that, $\sigma_{G'}(v_5, v_6) = \frac{3}{\sqrt{5\times 3}} = 0.77$, so v_6 rejoins $N_{G'}^\epsilon[v_5]$.

C. Global Selection

The concept of promotion cost is the basis for solving Subproblem 2 (global selection). For a target vertex t and some *fixed* vertex v, we determine the preferred strategy applicable to v by solving Subproblem 1. Using the determined strategy, we define the number of edges required to raise the similarity between v and t to at least ϵ as the **promotion cost** of v. In other words, when using the preferred strategy, the similarity between v and t can be made to at least ϵ after inserting edges whose required number equals the promotion cost.

The key to resolving Subproblem 2 is to choose one vertex at a time with the lowest promotion cost (i.e., the number of required edges to be inserted), until the budget is used up. In this way, given a budget k, we are likely to make a large number of vertices join t's ϵ-neighborhood.

The next question is how to compute the promotion cost of each $v \in V$. Recall that strategies S1 and S2 are candidates for the preferred strategy (since other strategies are less effective and require more edges). We thus present the costs of using S1 and S2 to promote v in Lemma 8 and Lemma 9, respectively.

Lemma 8. *Given a vertex $v \notin N_G^\epsilon[t]$, it requires $c^{s1}(v)$ edges to be inserted to make $\sigma_{G'}(t,v) \geq \epsilon$ by S1, where $b = \epsilon^2 d_G[t] - 2d_G[t,v]$, $c = 4\epsilon^2 d_G[t](d_G[v] - d_G[t,v]) + \epsilon^4 d_G[t]^2$, and*

$$c^{s1}(v) = \begin{cases} \lceil \frac{b-\sqrt{c}}{2} \rceil, & \text{if } \frac{b+\sqrt{c}}{2} \geq \frac{b-\sqrt{c}}{2} > 0; \\ \lceil \frac{b+\sqrt{c}}{2} \rceil, & \text{if } \frac{b+\sqrt{c}}{2} \geq 0 \geq \frac{b-\sqrt{c}}{2}; \\ -1, & \text{otherwise.} \end{cases} \quad (1)$$

Lemma 9. *Given a vertex $v \notin N_G^\epsilon[v]$, it requires $c^{s2}(v)$ edges to be inserted to make $\sigma_{G'}(t,v) \geq \epsilon$ by S2, where $b = \epsilon^2 d_G[v] - 2d_G[t,v]$, $c = 4\epsilon^2 d_G[v](d_G[t] - d_G[t,v]) + \epsilon^4 d_G[v]^2$, and*

$$c^{s2}(v) = \begin{cases} \lceil \frac{b-\sqrt{c}}{2} \rceil, & \text{if } \frac{b+\sqrt{c}}{2} \geq \frac{b-\sqrt{c}}{2} > 0; \\ \lceil \frac{b+\sqrt{c}}{2} \rceil, & \text{if } \frac{b+\sqrt{c}}{2} \geq 0 \geq \frac{b-\sqrt{c}}{2}; \\ -1, & \text{otherwise.} \end{cases} \quad (2)$$

Example 14. Consider the graph G in Fig. 1. Suppose we want to use S1 to improve the similarity between $t = v_5$ and $v = v_8$ to at least $\epsilon = 0.7$. Since $b = \epsilon^2 d_G[t] - 2d_G[t,v] = -2.04$; $c = 4\epsilon^2 d_G[t](d_G[v] - d_G[t,v]) + \epsilon^4 d_G[t]^2 = 19.52$, we get $\frac{b-\sqrt{c}}{2} = -3.23$ and $\frac{b+\sqrt{c}}{2} = 1.19$, which gives a cost of $2 = \lceil 1.19 \rceil$. After inserting 2 edges in the graph by S1, $\sigma_{G'}(v_5, v_8) = \frac{4}{\sqrt{4\times 6}} = 0.82 \geq \epsilon$, i.e., the similarity requirement is satisfied.

Algorithm 3: MNS

Input: graph $G(V, E)$, parameter ϵ, budget k, target t
Output: edges $\overline{E}$

1 **while** $k > 0$ **do**
2 $Q \leftarrow$ an empty min-priority queue;
3 **for** *each vertex* $v \in V$ **do**
4 **if** $v \in N_G^{\epsilon}[t]$ **then** continue;
5 $\overline{E}(v) \leftarrow \emptyset, T \leftarrow \emptyset$;
6 **if** $v \notin N_G[t]$ **then** $\overline{E}(v) \leftarrow$ Algorithm 2;
  `// Strategy S1`
7 **if** $d_G[t] \leq d_G[v]$ **then**
8 compute $c^{s1}(v)$ by Equation 1;
9 **if** $c^{s1}(v) < 0$ or $c^{s1}(v) > |\{N[t-v] \setminus t\}|$ **then** continue;
10 $T \leftarrow$ select $c^{s1}(v)$ vertices from $\{N[t-v] \setminus t\}$;
11 $\overline{E}(v) \leftarrow \overline{E}(v) \cup \{(v,u)|u \in T\}$;
12 **if** $|\overline{E}(v)| \leq k$ **then** insert $v \rightarrow Q$ with key $|\overline{E}(v)|$;
13 **else**
  `// Strategy S2`
14 compute $c^{s2}(v)$ by Equation 2;
15 **if** $c^{s2}(v) < 0$ or $c^{s2}(v) > |\{N[v-t] \setminus v\}|$ **then** continue;
16 $T \leftarrow$ select $c^{s2}(v)$ vertices from $\{N[v-t] \setminus v\}$;
17 $\overline{E}(v) \leftarrow \overline{E}(v) \cup \{(t,u)|u \in T\}$;
18 $E_T \leftarrow$ Algorithm 1, $\overline{E}(v) \leftarrow \overline{E}(v) \cup E_T$;
19 **if** $|\overline{E}(v)| \leq k$ **then** insert $v \rightarrow Q$ with key $|\overline{E}(v)|$;
20 **if** $Q = \emptyset$ **then** break;
21 $v \leftarrow$ pop from Q, $\overline{E} \leftarrow \overline{E} \cup \overline{E}(v)$, $k \leftarrow k - |\overline{E}(v)|$;
22 insert $\overline{E}(v)$ into G;
23 **return** $\overline{E}$;

D. Overall Algorithm

We now present the algorithm, denoted as MNS, for solving the MN problem. MNS operates in rounds and chooses the lowest cost vertex in each round. To obtain the lowest cost vertex, MNS (1) first applies a degree-based technique to identify the preferred strategy for each vertex v (by solving Subproblem 1); (2) then uses Equation 1 or Equation 2 to calculate the promotion cost for each v based on the identified strategy (by solving Subproblem 2). This process continues until the budget is exhausted.

MNS is described in Algorithm 3. We iteratively select a vertex in reach round, until the remaining budget k is less than zero (Line 1), or there are no candidates to choose from (Line 20). In each round, for each $v \in V$ (Line 3), we skip v if v is already in t's ϵ-neighborhood (Line 4). Next, if $v \notin N_G[t]$, we use Algorithm 2 as preprocessing to connect t to v, and then we place the returned edges of Algorithm 2 to $\overline{E}(v)$ (Line 6). Following that, we divide into two cases depending on the degree of v.

$d_G[t] \leq d_G[v]$. We use S1 (Line 7-12). We first compute $c^{s1}(v)$ using Equation 1 (Line 8), and skip if $c^{s1}(v)$ is negative or the size of $\{N[t-v] \setminus t\}$ is smaller than $c^{s1}(v)$ (Line 9). Otherwise, we select $c^{s1}(v)$ vertices T from $\{N[t-v] \setminus t\}$ (Line 10), and merge edges $\{(v,u)|u \in T\}$ into $\overline{E}(v)$ (Line 11). v is added into queue Q only if the number $|\overline{E}(v)|$ does not exceed the remaining budget k (Line 12).

$d_G[t] > d_G[v]$. We use S2 (Line 13-19). The processing logic is similar to the above case, with the main difference being in Line 18. Due to the side effect of using S2, we need to call Algorithm 1 to find edges E_T to eliminate the side effect. We merge E_T into $\overline{E}(v)$ as the required edges to promote v.

In both cases, we pop the lowest cost vertex v from Q as the selected vertex for this round, and subtract the cost $|\overline{E}(v)|$ of v from the budget k (Line 21). Note that we need to insert $\overline{E}(v)$ into graph G when proceeding to the next round (Line 22). We omit how to maintain the vertex degree when the graph changes, since this operation can be easily embedded in the algorithm.

Theorem 6. *The time cost of Algorithm 3 is* $O(k \cdot |V| \cdot |N_G^{\epsilon}[t]| \cdot d_{\max})$, *where* $d_{\max}$ *is the maximum degree of vertices in* G.

V. Experiments

A. Settings

Datasets. We conducted experiments on six real-world graphs to validate the effectiveness of our proposed algorithm MNS in solving the MN problem. These graphs were downloaded from Stanford Large Network Dataset Collection [41]. Table I gives the details of the graphs, including the node number (n), edge number (m) of each graph.

TABLE I: Description of Datasets

Abbr.	Dataset	n	m
LAST	lastfm	7,624	27,806
DEEZ	deezer-Eu	28,281	92,752
WIKI	wiki-vote	7,115	100,762
TWIT	twitch	9,498	153,138
FACE	facebook	22,470	170,823
GITH	github	37,700	289,003

Algorithms. As the MN problem has not been studied before, we cannot find a comparison method. To test the proposed algorithm MNS, we designed the following four baselines for comparison.

- DEG. DEG is similar to MNS in that it also selects one vertex per round to join the ϵ-neighborhood of the target vertex t. Specifically, DEG uses the same degree-based technique as MNS to determine the preferred strategy for each vertex (solving Subproblem 1). In this way, we know how to insert edges to improve the similarity between each vertex and t to at least ϵ. However, when solving Subproblem 2, DEG determines the selected vertices for each round directly based on the degree, without taking the promotion cost into account. In other words, DEG prefers vertices with a large degree for similarity promotion.
- SIM. SIM is similar to MNS in solving Subproblem 1, but when solving Subproblem 2, SIM prefers vertices that have a large similarity to the target vertex. SIM first calculates the similarity of all vertices to the target vertex, and then chooses the vertex with high similarity to the target vertex for similarity promotion in preference.
- kDEG. Rather than selecting vertices in rounds as MNS to determine the inserted edges, kDEG inserts edges between the target vertex and the k (i.e., budget) vertices with the highest degree in the graph.
- kSIM. Like kDEG, kSIM inserts edges between the target vertex and the k (i.e., budget) vertices in the graph that are most similar to the target vertex.

All algorithms were implemented in C++ and compiled with GNU GCC 4.8.5 with -O3 level optimization. All experiments were conducted on a machine with Intel Xeon 2.5 GHz CPU and 768 GB memory running Linux (Red Hat Linux 4.8.5, 64 bit).

Parameters. The definition of ϵ-neighborhood contains a parameter ϵ, which is a similarity threshold taking values between 0 and 1. We set the value of ϵ to 0.3, 0.4, 0.5, 0.6, 0.7, and 0.8. As suggested by recent work on SCAN [4], the default value of ϵ is chosen to be 0.6. Besides, our algorithm has a parameter, budget k, which is the number of edges allowed to be inserted. We set the value of k to 20, 40, 60, 80, and 100. The default value of k is taken as 60.

Metric. To measure the effectiveness of an algorithm, we use the amount of change in the target vertex's ϵ-neighborhood size

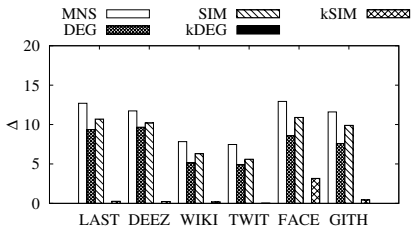


Fig. 6: The Comparison of Different Methods on the Improvement Amount

as an evaluation metric. Specifically, for a target vertex t, we compute the size $N^{\epsilon}_{G}[t]$ on graph G and the size $N^{\epsilon}_{G'}[t]$ on the graph G' after inserting edges. We define the *improvement* of t's ϵ-neighborhood size as $\Delta[t] = N^{\epsilon}_{G'}[t] - N^{\epsilon}_{G}[t]$. A higher value of $\Delta[t]$ implies that the method is more effective for the MN problem. To fairly compare the effectiveness of different algorithms, we randomly select 30 vertices whose initial ϵ-neighborhood size is 1 from the graph as target vertices. We record the improvement amount of each target and report the average improvement of these 30 vertices. If not specified, we use the average improvement as the evaluation criterion.

B. Results

Ex-1: Comparison of Different Methods. To compare the proposed method MNS with the baselines (DEG, SIM, kDEG and kSIM), we set the parameters to their default values (i.e., $\epsilon = 0.6$, $k = 60$). The improvement amount of all methods is calculated, and the results are shown in Fig. 6. We have the following findings.

- *Comparison with* DEG *and* SIM. On all used graphs, our method MNS outperforms DEG and SIM. The improvement of MNS is on average 1.44 times that of DEG and 1.21 times that of SIM. The difference between our method and DEG and SIM is how to solve Subproblem 2, i.e., how to select vertices for similarity promotion. Our method outperforms others by selecting vertices with the lowest promotion cost in each round.
- *Comparison with* kDEG *and* kSIM. The performance of kDEG and kSIM is poor: (1) the improvement amount of kDEG is 0 on all used graphs; (2) While kSIM has some improvement, the improvement amount of our method MNS is 66.62 times that of kSIM. kDEG and kSIM both solve the MN problem by inserting edges directly instead of selecting vertices. This shows that simply inserting edges does not solve the MN problem effectively, which verifies the necessity of our proposed method.

Ex-2: Effect of Budget k. We set the default value of the budget k to 60. To test the effect of the budget on the improvement amount, we vary the budget values from 20, 40, 60, 80, to 100. For each budget value, we set ϵ to the default value of 0.6. Since kDEG and kSIM do not perform well, we will not report their results in the following experiments. We report in Fig. 7 the results of three methods (MNS, DEG, and SIM) and have the following findings.

- *The improvement amount for all methods grows as k increases.* We find that the improvement amount of all methods increases with the increase of k on all graphs. This is reasonable because all methods pay a certain promotion cost per round to select some vertex, and thus the number of vertices selected (i.e., the improvement amount) is constrained by the budget. As the budget increases, more vertices can be selected to join the ϵ-neighborhood of the target vertex. This also demonstrates that we indeed bypass the MN's non-monotonicity nature.
- MNS *performs the best at all budgets.* MNS outperforms all other methods on all graphs and at all budget values. This shows that MNS maintains a good performance under different budgets. Also, the benefit of our method grows when k is increased. For example, on DEEZ, when k is varied from 20, 40, 60, 80, to 100, the improvement of MNS is 1.1, 1.16, 1.24, 1.27, and 1.32 times that of SIM, respectively.

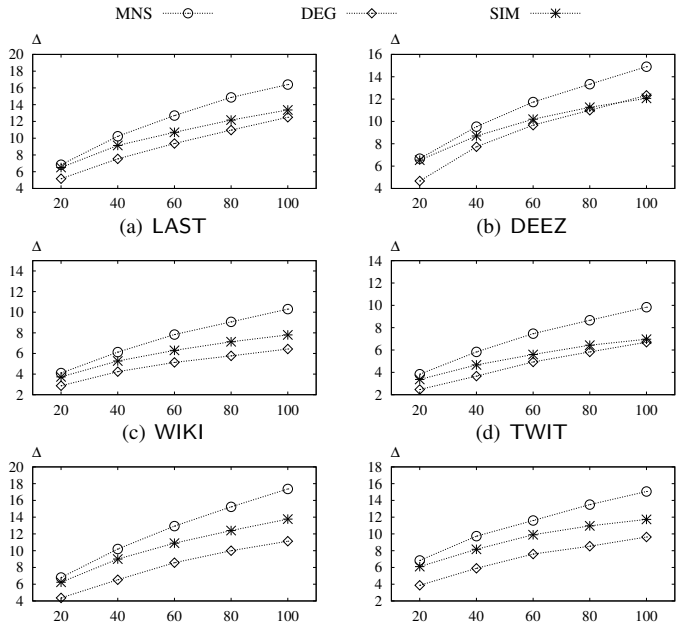


Fig. 7: The Effect of Budget k on the Improvement Amount

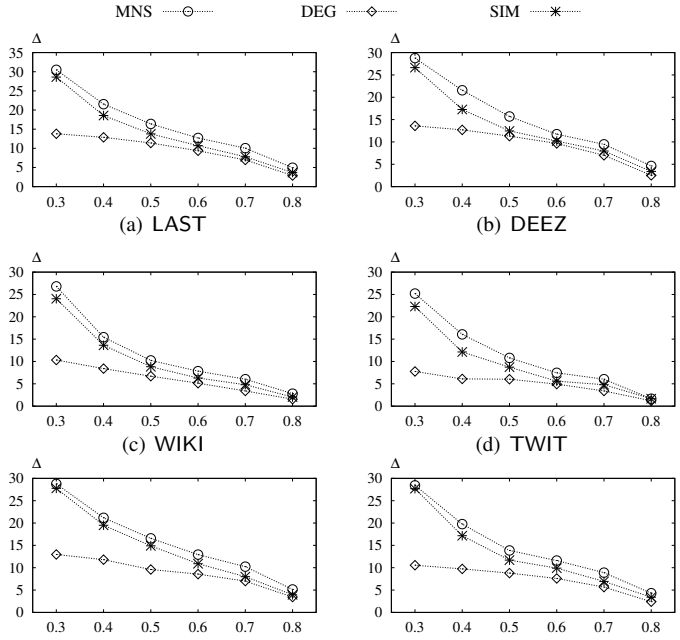


Fig. 8: The Effect of Parameter ϵ on the Improvement Amount

Ex-3: Effect of Parameter ϵ. We set the default value of the parameter ϵ to 0.6. To check the effect of ϵ, we vary the ϵ values from 0.3, 0.4, 0.5, 0.6, 0.7, to 0.8. For each value of ϵ, we set the budget k to the default value of 60. The results are reported in Fig. 8 and yield the following findings.

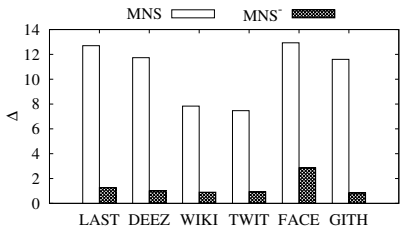


Fig. 9: The Effect of Side Effect Handling

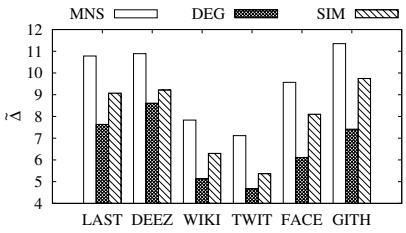


Fig. 11: The Test of the Relative Improvement Amount

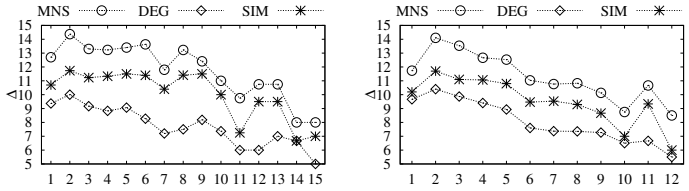


Fig. 10: The Effect of the Initial ϵ-Neighborhood Size

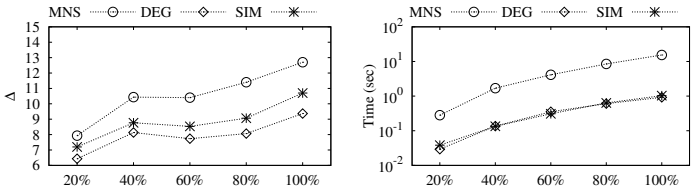


Fig. 12: The Test of the Scalability

- *The improvement amount for all methods decreases as ϵ increases.* On all graphs, the improvement amount for all methods decreases as the value of ϵ increases. Note that as the value of ϵ increases, more edges need to be inserted (i.e., at a higher promotion cost) to improve the similarity. The number of vertices that can join the ϵ-neighborhood then decreases for a given budget,.
- MNS *performs the best for all ϵ values.* Although the improvement of MNS decreases as ϵ rises, MNS still outperforms all other methods. This means that our method MNS is always a good choice when ϵ varies.

Ex-4: Effect of Side Effect Handling. When using strategy S2 and preprocessing (Algorithm 2), an increase in the target vertex t's degree can introduce the side effect for MNS. Algorithm 1 is proposed to address the side effect. To verify the necessity and effectiveness of the side effect handling technique, we remove Algorithm 1 from MNS to obtain the method MNS$^-$. We set k to 60 and ϵ to 0.6 to test the performance of MNS and MNS$^-$. The results are shown in Fig. 9, and we have the following finding.

- *The side effect handling technique is critical.* The improvement amount of MNS is on average 9.48 times greater than that of MNS$^-$. This result shows that side effect handling is necessary. Thanks to our handling technique, we can guarantee that when adding new vertices, other vertices already in t's ϵ-neighborhood will not leave.

Ex-5: Effect of Initial Size. In the previous experiments, the initial ϵ-neighborhood size of the target vertices was 1. To explore the effect of the initial ϵ-neighborhood size of target vertices, we obtain the initial sizes of all vertices in a graph and group the vertices based on their initial sizes. Then, we choose 30 vertices from *each* vertex group (with a specific initial size) as target vertices, and set the values of ϵ and budget k to their default values. Due to space constraints, we only show the results for the first two graphs (LAST and DEEZ) in Figure 10, where the maximum initial size of vertices in LAST is 15, and the maximum initial size of vertices in DEEZ is 12.

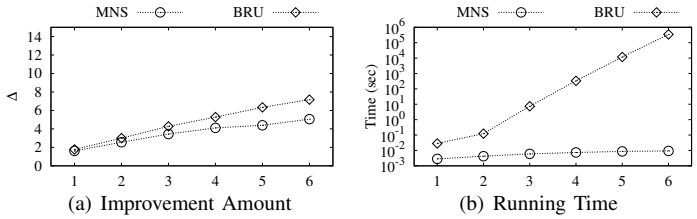
- MNS *performs the best with varying initial sizes.* Figure 10 shows that, for target vertices selected from vertex groups of different initial ϵ-neighborhood sizes, our method MNS outperforms other methods in terms of the improvement amount. For example, on LAST, if the initial size of the target vertices is 14, the improvement amount of MNS is 1.2 times that of DEG; if the initial size of the target vertices is 15, the improvement amount of MNS is 1.6 times that of DEG. This shows that the superiority of our method is not affected by the initial size.

Ex-6: Test of Relative Improvement Amount. To further investigate the effect of target vertex selection, 30 vertices are chosen at *random* as target vertices. Furthermore, a new metric, relative improvement amount, is proposed to assess the performance of various methods. The relative improvement amount $\tilde{\Delta}[t]$ of a target vertex t is defined as the ratio of the improvement amount $\Delta[t]$ to the initial ϵ-neighborhood size of t, i.e., $\tilde{\Delta}[t] = \frac{\Delta[t]}{|N_G^{\epsilon}[t]|}$. The parameters ϵ and budget k are set to their default values, and the average relative improvement amount of these target vertices is shown in Fig. 11. We have the following finding.

- *The relative improvement amount of* MNS *is the best.* On the six graphs used, the relative improvement amount of MNS is on average 1.47 times that of DEG and 1.21 times that of SIM. This indicates that MNS outperforms the other methods when target vertices are chosen randomly. On the other hand, the above results (using the relative improvement amount as the criterion) are similar to those in Ex-1 (using the improvement amount as the criterion), indicating that MNS performs well under both evaluation criteria.

Ex-7: Test of Scalability. In testing scalability, we divide the vertices of a graph into five disjoint groups, each group containing 20% of the graphs' vertices. We generate five test subgraphs for this graph, where the i-th subgraph contains the vertices (and the edges between them) from the first i groups. For each test subgraph, we set the parameter ϵ and budget k to default values and randomly select 30 vertices as target vertices. We count the improvement amount and the running time on each test subgraph. Due to space constraints, we only report the results on LAST in Fig. 12, and obtain the following findings.

- *On effectiveness.* Since the target vertices are chosen randomly, the improvement amount of various methods does not always increase with the vertex number. However, MNS outperforms the other methods on different test subgraphs. For example, on the subgraph with 60% vertices, the improvement amount of



(a) Improvement Amount (b) Running Time

Fig. 13: The Comparison with the Exact Solution

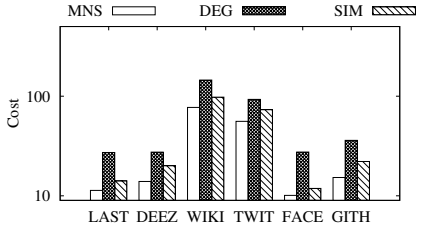


Fig. 14: The Comparison for the Dual Problem

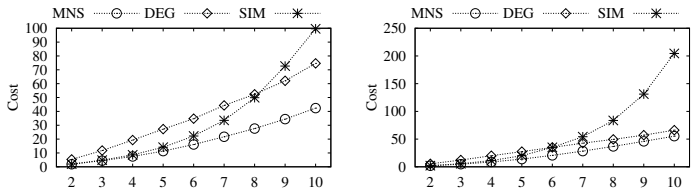
MNS is 1.34 times that of DEG and 1.22 times that of SIM. On the subgraph with 80% vertices, the improvement amount of MNS is 1.41 times that of DEG and 1.26 times that of SIM.

- *On efficiency.* The running time of DEG and SIM is better than that of MNS: on the five test subgraphs, the runtime of MNS is on average 12.89 times that of DEG and 12.33 times that of SIM. However, MNS can finish the computation within 16 seconds on all test subgraphs. This suggests that by sacrificing some runtime (but still runs in seconds), MNS can achieve better effectiveness (regarding the improvement amount) than DEG and SIM. This sacrifice is desirable for the MN problem where the optimization objective (i.e., the improvement amount) is discrete. For example, suppose SCAN calls a vertex a core vertex when its ϵ-neighborhood size reaches a discrete value μ. Given a budget, assume that MNS increases the size of a vertex to μ, but DEG and SIM can only increase the size to $\mu-1$. In this case, although MNS has only one more improvement amount than the other methods, MNS succeeds in changing the role of this vertex to a core vertex, while the other methods fail.

Ex-8: Comparison with Exact Solution. We compare our method MNS with the brute force (exact) method (denoted as BRU). BRU enumerates all combinations of (up to) k edges, where k is the budget, to obtain the (maximum possible) improvement amount of a certain target vertex. Since BRU cannot complete the computation on the graphs listed in Table I in a reasonable time, the small graph SOUT[3] [42], [43] is used. SOUT is an interaction network with 18 vertices and 64 edges. We change the budget k from 1 to 6 and use each vertex as a target vertex to count the average improvement amount as well as the running time. The results are presented in Fig. 13.

- *On effectiveness.* BRU is an exact algorithm, so its improvement amount must be greater than that of MNS. However, the difference is limited: when budget k is varied from 1 to 6, the improvement amount of BRU is on average 1.28 times that of MNS.
- *On efficiency.* When k is changed from 1 to 6, the running time of BRU is 10.08, 29.2, 1252.51, 45207.66, 1354701.93, and 37326338.56 times that of MNS, respectively. In particular, when k is 6, BRU takes over 4 days to finish, whereas MNS takes less than 0.01 seconds. Because of the long computation time, BRU is infeasible in practice. In contrast, our method guarantees a short running time by sacrificing some improvement amount (but still within 1.28 times of the exact algorithm).

[3]https://networkrepository.com/ia-southernwomen.php



(a) LAST (b) DEEZ

Fig. 15: The Effect of Parameter μ for the Dual Problem

C. Results of Dual Problem

The MN problem, which is to maximize the target vertex t's ϵ-neighborhood size with a given budget k, is the focus of this paper. As a dual of the MN problem, we want to minimize the number of inserted edges (i.e., the **total cost**) so that the target vertex has a specific neighborhood size μ. The significance of this dual problem is that we can know how to convert non-core vertices (e.g., outlier vertices) to core vertices with the fewest edges. In practice, SCAN can be used for anomaly detection, which works by identifying outlier vertices as anomalies [44]–[46]. Solving the dual problem allows us to assess the robustness of SCAN for anomaly detection.

Our proposed method MNS, as well as baselines like DEG and SIM, can be easily adapted to solve this dual problem. The basic idea is that we still choose one vertex per round, but the stopping condition shifts from the total cost exceeding the budget to t's ϵ-neighborhood size reaching the required μ. For the dual problem, we use the total cost at stopping as an evaluation metric.

Ex-9: Comparison Between Different Methods. To compare the performance of various methods for solving this dual problem, we set the value of ϵ to 0.6 and the value of μ to 5, as suggested by recent work in SCAN [4]. To fairly compare different methods, we randomly selected 30 target vertices whose ϵ-neighborhood size is 1 in the graph. The total cost is averaged across the 30 vertices, and the results are shown in Fig. 14. We discover the following.

- MNS *outperforms the baselines.* On the graphs used, DEG requires on average 2.17 times the cost of our method MNS and SIM requires 1.31 times the cost of MNS. Note that a low cost means that a method is more effective. These results show that our method MNS outperforms the baselines in solving the dual problem of MN.

Ex-10: Effect of Parameter μ. In Ex-9, we set μ to 5. To see the effect of μ, we varied the value of μ from 2, 5, 10, 12 to 15. Due to space constraints, we only report results on the first two graphs (LAST and DEEZ) in Fig. 15. Similar results are observed on the other graphs, and we have the following finding.

- *The cost of all methods increases as μ varies.* When μ rises, the cost needed for all methods (MNS, DEG, and SIM) increases

as well. This is reasonable because as μ grows, we have to add more vertices to the target vertex's ϵ-neighborhood, resulting in higher costs.

- MNS *performs the best for all μ values.* Although the cost of MNS increases as ϵ rises, MNS outperforms the other methods for all values of μ. This means that MNS's superiority in solving the dual problem is not affected by μ.

VI. Conclusion

We present and solve the MN problem in this paper. We first demonstrate the challenge of the MN problem by showing that it is both NP-hard and APX-hard. Furthermore, we show that this problem is neither submodular nor monotonic. To bypass this problem's non-monotonicity nature, we switch from enumerating edges to selecting vertices, and identify two key subproblems: local promotion and global selection. By solving these subproblems, we obtain the algorithm to solve the MN problem. The experimental results on real graphs validate the effectiveness of our algorithm. Our subsequent work includes considering more factors in defining the MN problem and designing more effective and efficient algorithms for solving the MN problem.

Appendix

Proof of Lemma 1. Suppose l edges are inserted into G, then $\sigma_{G'}(t,v) = \frac{d_{G'}[t,v]}{\sqrt{d_{G'}[t]\times d_{G'}[v]}} = \frac{d_G[t,v]+l}{\sqrt{d_G[t]\times(d_G[v]+l)}}$ since

1) $d_{G'}[t,v] = d_G[t,v] + l$ as l vertices in $N_G[t-v]$ become the shared neighbors of t and v in G';
2) $d_{G'}[v] = d_G[v] + l$ as v connects to l vertices in $N_G[t-v]$;
3) $d_{G'}[t] = d_G[t]$ as no edge is attached to t.

We then prove that $\sigma_{G'}(t,v) > \sigma_G(t,v)$.

$$\frac{\sigma_{G'}(t,v)}{\sigma_G(t,v)} = \frac{\frac{d_G[t,v]+l}{\sqrt{d_G[t]\times(d_G[v]+l)}}}{\frac{d_G[t,v]}{\sqrt{d_G[t]\times d_G[v]}}} = \sqrt{\frac{(1+\frac{l}{d_G[t,v]})^2}{1+\frac{l}{d_G[v]}}} \geq \sqrt{\frac{(1+\frac{l}{d_G[v]})^2}{1+\frac{l}{d_G[v]}}} > 1.$$

Proof of Lemma 2. Similar to the proof of Lemma 1.

Proof of Lemma 3. Suppose x ($0 \leq x \leq l$) edges are chosen from $\{(v,w)|w \in N_G[t-v]\}$, and $l-x$ edges are from $\{(t,w)|w \in N_G[v-t]\}$, then

1) $d_{G'}[t,v] = d_G[t,v] + l$ as l new common neighbors are formed, x from $N_G[t-v]$ and $l-x$ from $N_G[v-t]$;
2) $d_{G'}[v] = d_G[v] + x$ as v connects to x vertices in $N_G[t-v]$;
3) $d_{G'}[t] = d_G[t] + (l-x)$ as t connects to $l-x$ vertices in $N_G[v-x]$.

The similarity in G' is $\sigma_{G'}(t,v) = \frac{d_G[t,v]+l}{\sqrt{(d_G[t]+l-x)(d_G[v]+x)}}$.

Since the numerator $d_G[t,v]+l$ is the same for all assignment of x, to maximize $\sigma_{G'}(t,v)$, we need to minimize the denominator $f(x) = (d_G[t]+l-x)(d_G[v]+x)$. We take the first and second order derivatives of $f(x)$: $f'(x) = -2x + (d_G[t] - d_G[v] + l)$, $f''(x) = -2$. This means $f(x)$ takes the minimum value when $x = l$ (choosing l edges from $\{(v,w)|w \in N_G[t-v]\}$) or $x = 0$ (choosing l edges from $\{(t,w)|w \in N_G[v-t]\}$).

Proof of Lemma 4. By the proof of Lemma 3, $\sigma^{s1}_{G'}(t,v) = \frac{d_G[t,v]+l}{\sqrt{d_G[t]\times(d_G[v]+l)}}$ and $\sigma^{s2}_{G'}(t,v) = \frac{d_G[t,v]+l}{\sqrt{(d_G[t]+l)\times d_G[v]}}$.

We then derive the condition when $\sigma^{s1}_{G'}(t,v) > \sigma^{s2}_{G'}(t,v)$:

$$\because \quad \frac{\sigma^{s1}_{G'}(t,v)}{\sigma^{s2}_{G'}(t,v)} = \frac{\frac{d_G[t,v]+l}{\sqrt{d_G[t]\times(d_G[v]+l)}}}{\frac{d_G[t,v]+l}{\sqrt{(d_G[t]+l)\times d_G[v]}}} = \sqrt{\frac{(d_G[t]+l)\times d_G[v]}{d_G[t]\times(d_G[v]+l)}} > 1$$
$$\therefore \quad (d_G[t]+l)\times d_G[v] > d_G[t]\times(d_G[v]+l).$$

And it follows that $d_G[t] < d_G[v]$. Similarly, we can deduce that $\sigma^{s1}_{G'}(t,v) < \sigma^{s2}_{G'}(t,v)$ yields $d_G[t] > d_G[v]$.

Proof of Lemma 5. $\sigma^{s3}_{G'}(t,v) = \frac{d_G[t,v]+\frac{l}{2}}{\sqrt{(d_G[t]+\frac{l}{2})\times(d_G[v]+\frac{l}{2})}}$ since

1) $d_{G'}[t,v] = d_G[t,v] + \frac{l}{2}$ as $\frac{l}{2}$ vertices in $\{V \setminus \{N[v] \cup N[t]\}$ become the neighbors of both t and v in G';
2) $d_{G'}[v] = d_G[v] + \frac{l}{2}$ as v connects to $\frac{l}{2}$ vertices;
3) $d_{G'}[t] = d_G[t] + \frac{l}{2}$ as v connects to $\frac{l}{2}$ vertices.

We prove $\sigma^{s1}_{G'}(t,v) = \frac{d_G[t,v]+l}{\sqrt{d_G[t]\times(d_G[v]+l)}} > \sigma^{s3}_{G'}(t,v)$ and the case for $\sigma^{s2}_{G'}(t,v) = \frac{d_G[t,v]+l}{\sqrt{d_G[t]\times(d_G[v]+l)}} > \sigma^{s3}_{G'}(t,v)$ is similar.

$$\begin{aligned}\frac{\sigma^{s1}_{G'}(t,v)}{\sigma^{s3}_{G'}(t,v)} &= \frac{d_G[t,v]+l}{d_G[t,v]+\frac{l}{2}} \times \sqrt{\frac{d_G[t]\times(d_G[v]+l)}{(d_G[t]+\frac{l}{2})\times(d_G[v]+\frac{l}{2})}} \\ &> \frac{d_G[t,v]+l}{d_G[t,v]+\frac{l}{2}} \times \sqrt{\frac{d_G[t]}{d_G[t]+\frac{l}{2}}} = \sqrt{\frac{(d_G[t,v]+l)^2\times d_G[t]}{(d_G[t,v]+\frac{l}{2})^2\times(d_G[t]+\frac{l}{2})}}.\end{aligned}$$

We claim the above equation is greater than 1 because:

$$\begin{aligned}& (d_G[t,v]+l)^2 \times d_G[t] - (d_G[t,v]+\tfrac{l}{2})^2 \times (d_G[t]+\tfrac{l}{2}) \\ =\ & (d_G[t,v]\times d_G[t]\times l - d_G[t,v]^2\times l) + (\tfrac{3}{4}d_G[t]\times l^2 - \tfrac{1}{8}l^3) \\ >\ & 0 \qquad\qquad \because\ d_G[t] \geq d_G[t,v], d_G[t] \geq l.\end{aligned}$$

Proof of Lemma 6. $\sigma_{G'}(t,u) = \frac{d_G[t,u]}{\sqrt{(d_G[t]+|T|)\times d_G[u]}} < \sigma_{G'}(t,u) = \frac{d_G[t,u]}{\sqrt{d_G[t]\times d_G[u]}}$ if t connects to the vertices in T, since

1) $d_{G'}[t,u] = d_G[t,u]$ as $N_G[u] \cap T = \emptyset$;
2) $d_{G'}[u] = d_G[u]$ as no vertex connects to u;
3) $d_{G'}[t] = d_G[t] + |T|$ as t connects to vertices in T.

Proof of Lemma 7. Before inserting l edges, $\sigma_G(t,u) = \frac{d_G(t,u)}{\sqrt{d_G[t]\times d_G[u]}} \geq \epsilon$ by definition. When l edges are added between t and T, $\sigma_{G'}(t,u) \geq \frac{d_G(t,u)+l}{\sqrt{(d_G[t]+l)\times(d_G[u]+l)}}$ since

1) $d_{G'}[t,u] = d_G[t,u]+l$ as l common neighbors (T) are formed;
2) $d_{G'}[u] \leq d_G[u] + l$ as T and $N_G[u]$ may overlap;
3) $d_{G'}[t] = d_G[t] + l$ as l edges are added between t and T.

We then prove that $\sigma_{G'}(t,u) \geq \sigma_G(t,u) \geq \epsilon$.

$$\begin{aligned}\frac{\sigma_{G'}(t,u)}{\sigma_G(t,u)} &= \frac{d_G[t,u]+l}{d_G[t,u]} \times \sqrt{\frac{d_G[t]\times d_G[u]}{(d_G[t]+l)\times(d_G[u]+l)}} \\ &= \sqrt{\frac{(d_G[u,t]+l)^2\times d_G[t]\times d_G[u]}{d_G[u,t]^2\times(d_G[t]+l)\times(d_G[u]+l)}}.\end{aligned}$$

We claim the above equation is greater than 1 because:

$$\begin{aligned}& (d_G[t,v]+l)^2 \times d_G[t] - (d_G[t,v]+\tfrac{l}{2})^2 \times (d_G[t]+\tfrac{l}{2}) \\ =\ & (d_G[t,v]\times d_G[t]\times l - d_G[t,v]^2\times l) + (\tfrac{3}{4}d_G[t]\times l^2 - \tfrac{1}{8}l^3) \\ >\ & 0 \qquad\qquad \because\ d_G[t] \geq d_G[t,v], d_G[t] \geq l.\end{aligned}$$

Proof of Theorem 4. The similarity of any vertex pair can be computed in $O(d_{\max})$ [4]. For $\forall u \in N^{\epsilon}_G[t]$, we compute its similarity with t at most $|T|$ times.

Proof of Lemma 8. To make $\sigma_{G'}(t,v) = \frac{d_G[t,v]+l}{\sqrt{d_G[t]\times(d_G[v]+l)}} = \epsilon$, we solve Equation 3.

$$l^2 + (2d_G[t,v] - \epsilon^2 d_G[t])\times l + (d_G[t,v]^2 - \epsilon^2 d_G[t]\times d_G[v]) = 0. \quad (3)$$

Equation 3 always has a solution since:

$$\begin{aligned}c &= (2d_G[t,v] - \epsilon^2 d_G[t])^2 - 4\times(d_G[t,v]^2 - \epsilon^2 d_G[t]\times d_G[v]) \\ &= 4\epsilon^2 d_G[t](d_G[v] - d_G[t,v]) + \epsilon^4 d_G[t]^2 \geq 0.\end{aligned}$$

And the solutions are $\frac{b\pm\sqrt{c}}{2}$, where $b = \epsilon^2 d_G[t] - 2d_G[t,v]$.

Proof of Theorem 6. MNS works for at most k rounds. It scans at most $|V|$ vertices per round. For $\forall v \in V$, it takes $O(|N^{\epsilon}_G[t]| \cdot d_{\max})$ to handle the side effect and $O(d_{\max})$ to obtain the cost by Equations 1-2.

REFERENCES

[1] H. Shiokawa, Y. Fujiwara, and M. Onizuka, "Scan++ efficient algorithm for finding clusters, hubs and outliers on large-scale graphs," *Proceedings of the VLDB Endowment*, vol. 8, no. 11, pp. 1178–1189, 2015.

[2] R. Angles and C. Gutierrez, "Survey of graph database models," *ACM Computing Surveys (CSUR)*, vol. 40, no. 1, pp. 1–39, 2008.

[3] R. Angles, "A comparison of current graph database models," in *2012 IEEE 28th International Conference on Data Engineering Workshops*. IEEE, 2012, pp. 171–177.

[4] L. Chang, W. Li, L. Qin, W. Zhang, and S. Yang, "pscan: Fast and exact structural graph clustering," *IEEE Transactions on Knowledge and Data Engineering*, vol. 29, no. 2, pp. 387–401, 2017.

[5] S. E. Schaeffer, "Graph clustering," *Computer science review*, vol. 1, no. 1, pp. 27–64, 2007.

[6] B. M. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Recommender systems for large-scale e-commerce: Scalable neighborhood formation using clustering," in *Proceedings of the fifth international conference on computer and information technology*, vol. 1. Citeseer, 2002, pp. 291–324.

[7] J. Shi and J. Malik, "Normalized cuts and image segmentation," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 22, no. 8, pp. 888–905, 2000.

[8] M. E. Newman and M. Girvan, "Finding and evaluating community structure in networks," *Physical review E*, vol. 69, no. 2, p. 026113, 2004.

[9] P. Jiang and M. Singh, "Spici: a fast clustering algorithm for large biological networks," *Bioinformatics*, vol. 26, no. 8, pp. 1105–1111, 2010.

[10] T. Tseng, L. Dhulipala, and J. Shun, "Parallel index-based structural graph clustering and its approximation," in *Proceedings of the 2021 International Conference on Management of Data*, 2021, pp. 1851–1864.

[11] B. Ruan, J. Gan, H. Wu, and A. Wirth, "Dynamic structural clustering on graphs," in *Proceedings of the 2021 International Conference on Management of Data*, 2021, pp. 1491–1503.

[12] X. Xu, N. Yuruk, Z. Feng, and T. A. Schweiger, "Scan: a structural clustering algorithm for networks," in *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2007, pp. 824–833.

[13] W. Zhao, V. Martha, and X. Xu, "Pscan: a parallel structural clustering algorithm for big networks in mapreduce," in *2013 IEEE 27th International Conference on Advanced Information Networking and Applications (AINA)*. IEEE, 2013, pp. 862–869.

[14] S. Lim, S. Ryu, S. Kwon, K. Jung, and J.-G. Lee, "Linkscan*: Overlapping community detection using the link-space transformation," in *2014 IEEE 30th international conference on data engineering*. IEEE, 2014, pp. 292–303.

[15] H. Shiokawa, T. Takahashi, and H. Kitagawa, "Scalescan: scalable density-based graph clustering," in *International Conference on Database and Expert Systems Applications*. Springer, 2018, pp. 18–34.

[16] M.-S. Kim and J. Han, "A particle-and-density based evolutionary clustering method for dynamic networks," *Proceedings of the VLDB Endowment*, vol. 2, no. 1, pp. 622–633, 2009.

[17] H. Sun, J. Huang, J. Han, H. Deng, P. Zhao, and B. Feng, "gskeletonclu: Density-based network clustering via structure-connected tree division or agglomeration," in *2010 IEEE International Conference on Data Mining*. IEEE, 2010, pp. 481–490.

[18] D. Wen, L. Qin, Y. Zhang, L. Chang, and X. Lin, "Efficient structural graph clustering: an index-based approach," *Proceedings of the VLDB Endowment*, vol. 11, no. 3, pp. 243–255, 2017.

[19] X. Jin, C. X. Lin, J. Luo, and J. Han, "Socialspamguard: A data mining-based spam detection system for social media networks," *Proceedings of the VLDB Endowment*, vol. 4, no. 12, pp. 1458–1461, 2011.

[20] W. Li, M. Gao, F. Wu, W. Rong, J. Wen, and L. Qin, "Manipulating black-box networks for centrality promotion," in *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, 2021, pp. 73–84.

[21] G. Stringhini, C. Kruegel, and G. Vigna, "Detecting spammers on social networks," in *Proceedings of the 26th annual computer security applications conference*, 2010, pp. 1–9.

[22] L. Bilge, T. Strufe, D. Balzarotti, and E. Kirda, "All your contacts are belong to us: automated identity theft attacks on social networks," in *Proceedings of the 18th international conference on World wide web*, 2009, pp. 551–560.

[23] S. Webb, J. Caverlee, and C. Pu, "Social honeypots: Making friends with a spammer near you." in *CEAS*, 2008, pp. 1–10.

[24] A. E. Cinà, A. Torcinovich, and M. Pelillo, "A black-box adversarial attack for poisoning clustering," *Pattern Recognition*, vol. 122, p. 108306, 2022.

[25] B. Biggio, S. R. Bulò, I. Pillai, M. Mura, E. Z. Mequanint, M. Pelillo, and F. Roli, "Poisoning complete-linkage hierarchical clustering," in *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*. Springer, 2014, pp. 42–52.

[26] H. Dai, H. Li, T. Tian, X. Huang, L. Wang, J. Zhu, and L. Song, "Adversarial attack on graph structured data," in *International conference on machine learning*. PMLR, 2018, pp. 1115–1124.

[27] P. Dey and S. Medya, "Manipulating node similarity measures in networks," *arXiv preprint arXiv:1910.11529*, 2019.

[28] E. Rosenfeld, E. Winston, P. Ravikumar, and Z. Kolter, "Certified robustness to label-flipping attacks via randomized smoothing," in *International Conference on Machine Learning*. PMLR, 2020, pp. 8230–8241.

[29] B. Kadri, A. M'hamed, and M. Feham, "Secured clustering algorithm for mobile ad hoc networks," *International Journal of Computer Science and Network Security*, vol. 7, no. 3, pp. 27–34, 2007.

[30] J. Jia, B. Wang, X. Cao, and N. Z. Gong, "Certified robustness of community detection against adversarial structural perturbation via randomized smoothing," in *Proceedings of The Web Conference 2020*, 2020, pp. 2718–2724.

[31] A. Chakraborty, M. Alam, V. Dey, A. Chattopadhyay, and D. Mukhopadhyay, "A survey on adversarial attacks and defences," *CAAI Transactions on Intelligence Technology*, vol. 6, no. 1, pp. 25–45, 2021.

[32] W. Jin, Y. Li, H. Xu, Y. Wang, S. Ji, C. Aggarwal, and J. Tang, "Adversarial attacks and defenses on graphs," *ACM SIGKDD Explorations Newsletter*, vol. 22, no. 2, pp. 19–34, 2021.

[33] J. Crussell and P. Kegelmeyer, "Attacking dbscan for fun and profit," in *Proceedings of the 2015 SIAM International Conference on Data Mining*. SIAM, 2015, pp. 235–243.

[34] M. Ester, H.-P. Kriegel, J. Sander, X. Xu *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise." in *Kdd*, vol. 96, no. 34, 1996, pp. 226–231.

[35] A. Chhabra, A. Roy, and P. Mohapatra, "Suspicion-free adversarial attacks on clustering algorithms," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 04, 2020, pp. 3625–3632.

[36] B. Biggio, I. Pillai, S. Rota Bulò, D. Ariu, M. Pelillo, and F. Roli, "Is data clustering in adversarial settings secure?" in *Proceedings of the 2013 ACM workshop on Artificial intelligence and security*, 2013, pp. 87–98.

[37] Y. Chen, Y. Nadji, A. Kountouras, F. Monrose, R. Perdisci, M. Antonakakis, and N. Vasiloglou, "Practical attacks against graph-based clustering," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 1125–1142.

[38] A. Vasudeva and M. Sood, "Sybil attack on lowest id clustering algorithm in the mobile ad hoc network," *International Journal of Network Security & Its Applications*, vol. 4, no. 5, p. 135, 2012.

[39] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of computer computations*. Springer, 1972, pp. 85–103.

[40] U. Feige, "A threshold of ln n for approximating set cover," *Journal of the ACM (JACM)*, vol. 45, no. 4, pp. 634–652, 1998.

[41] J. Leskovec and A. Krevl, "SNAP Datasets: Stanford large network dataset collection," http://snap.stanford.edu/data, Jun. 2014.

[42] R. A. Rossi and N. K. Ahmed, "The network data repository with interactive graph analytics and visualization," in *AAAI*, 2015. [Online]. Available: https://networkrepository.com

[43] A. Davis, B. B. Gardner, and M. R. Gardner, "Deep south chicago," 1941.

[44] Z. Peng, M. Luo, J. Li, H. Liu, and Q. Zheng, "Anomalous: A joint modeling approach for anomaly detection on attributed networks." in *IJCAI*, 2018, pp. 3513–3519.

[45] L. Akoglu, H. Tong, and D. Koutra, "Graph based anomaly detection and description: a survey," *Data mining and knowledge discovery*, vol. 29, no. 3, pp. 626–688, 2015.

[46] R. Yu, H. Qiu, Z. Wen, C. Lin, and Y. Liu, "A survey on social media anomaly detection," *ACM SIGKDD Explorations Newsletter*, vol. 18, no. 1, pp. 1–14, 2016.